Journal Subline

LNCS 7776

Joanna Kołodziej
Tadeusz Burczyński
Marenglen Biba
Guest Editors

Transactions on Computational Collective Intelligence X

Ngoc Thanh Nguyen
Editor-in-Chief

Springer

Ngoc-Thanh Nguyen   Joanna Kołodziej
Tadeusz Burczyński   Marenglen Biba (Eds.)

Transactions on Computational Collective Intelligence X

Springer

Editor-in-Chief

Ngoc-Thanh Nguyen
Wrocław University of Technology
Wyb. Wyspiańskiego 27, 50-370 Wrocław, Poland
E-mail: ngoc-thanh.nguyen@pwr.edu.pl

Guest Editors

Joanna Kołodziej
Cracow University of Technology
Institute of Computer Science
ul. Warszawska 24, 31-155 Cracow, Poland
E-mail: jokolodziej@pk.edu.pl

Tadeusz Burczyński
Cracow University of Technology
Institute of Computer Science
ul. Warszawska 24, 31-155 Cracow, Poland
E-mail: tburczynski@pk.edu.pl

Marenglen Biba
University of New York in Tirana
Rr. Komuna e Parisit, Tirana, Albania
E-mail: marenglenbiba@unyt.edu.al

ISSN 0302-9743 (LNCS) e-ISSN 1611-3349 (LNCS)
ISSN 2190-9288 (TCCI)
ISBN 978-3-642-38495-0 e-ISBN 978-3-642-38496-7
DOI 10.1007/978-3-642-38496-7

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013938421

CR Subject Classification (1998): I.2.11, F.1.3, I.2.4, H.3.4, I.6.3-8, H.4.1-3, I.2.8

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Transactions on Computational Collective Intelligence – Vol. X

Special Issue on "Modelling and Simulation of Intelligent Large-Scale Systems"

Preface

Modeling and simulation are widely considered as essential tools in many areas of science and engineering for the prediction and analysis of complex systems and natural phenomena. They often require a significant amount of computational resources with large data sets typically scattered across different geographical locations. Furthermore, the development of such complex modeling and simulation environments usually requires collaborative efforts from researchers with different domain knowledge and expertise, possibly at distinct locations. Intelligent high-performance computing is arguably required to deal with the behavior and complexity of such large-scale systems.

Intelligent computing is usually defined as advanced computing methods and techniques based on classic computational intelligence, artificial intelligence, and intelligent agents. On the other hand, large-scale distributed systems—such as grids, peer-to-peer and ad-hoc networks, constellations, and clouds—enable the aggregation and sharing of highly distributed resources from different organizations with distinct owners, administrators, and policies. With the advent of such systems, where efficient inter-domain operation is one of the most important features, it is arguably required to investigate novel methods and techniques to enable secure access to data and resources, efficient scheduling, self-adaptation, decentralization, and self-organization. The concept of intelligent large-scale systems brings together results from all the above areas, making a positive impact on the development of new efficient models and systems simulators.

This special issue herewith presents 13 research papers with novel concepts in the analysis, implementation, and evaluation of the next generation of intelligent scalable techniques for data-intensive processing and global optimization problems in large-scale distributed systems.

The first five papers discuss novel scalable agent-based models and techniques for solving various data-intensive global optimization problems in large-scale computational environments. The presented techniques and their implementations are based on formal mathematical and logical models with the new semantic rules and modern synchronization modules of parallel computational processes. Byrski et al. present a theoretical model of immunnological multi-agent system (iMAS). This model is based on the general paradigms of homogeneous Markov chain theory. The authors demonstrate a Markov-based interpretation of the basic features of the IMAS system. An interesting practical approach of an intelligent system of software agents is presented by Redavid et al. in the second paper. The authors discuss the dynamic orchestration problem of Semantic Web services (SWSs). In their model, the orchestration capabilities are provided on-the-fly by the software agents that apply logical reasoning on SWSs descriptions. The most important feature of this model is dynamic adaptation to environment and system conditions (states). This problem has been explored by Piętak and Kisiel-Doronicki, who present a MAS framework for composition and implementation of the distributed computational by designing a proper configuration of the general MAS components. Agent-based models have become popular recently as efficient support systems in collective intelligence (CI). Li and Perera present a system with single chip hardware agents for emulation of collective intelligence models and deployment in realistic collaborative settings. Another simulation agent-based framework is presented by Ihrig. He developed a *SimISpace2* simulation environment designed to emulate the strategic knowledge management processes and knowledge-based agents' interactions.

The second part of this special issues consists of eight papers on recent models and developments in various types of large-scale distributed systems. One of the most important issues addressed by researchers in this domain is data storage and processing. Ichikawa and Uehara have developed a cloud search engine for various types of data stored online in infrastructure as a service (IaaS) cloud layer. This engine works as a distributed system composed of many local cooperated modules, which are responsible for reading the local documents in a storage server and creating their indexes, that are further sent to the global module. Kołodziej and Khan review in their paper the recent developments on data storage and processing during grid scheduling. They defined a generic unified model for access to data grid nodes and databases that is necessary for completing the scheduled tasks distributed in the grid environment. Another scalable cloud approach for improving the massive data processing during the execution of e-science application is presented by Terzo et al. The authors combine in their model a physical grid architecture with a virtual cloud service layer that guarantees flexibility and a progressive scalability of this hybrid system.

Rational utilization of the system nodes and conservation of the energy consumed during the computation and communication are the crucial issues in most high-performance distributed systems. Niewiadomska-Szynkiewicz defines energy-aware inter-node communication protocols in wireless sensor networks (WSNs). These protocols rely on hierarchical routing and periodic coordination technique in WSNs. The graphics processing unit (GPU) has the potential to enable a new generation of applications for small computational clusters. GPUs have demonstrated in many applications a cluster-level performance at a fraction of the cost and energy consumption of traditional CPUs for certain general-purpose applications. Garba et al. try to solve the multiple large Hermitian eigenvector and eigenvalue systems on GPUs. The recent research results and models in high-performed data mining are surveyed by Trandafili and Biba.

The last two papers address the decision-making problems in distributed environments. Boryczka et al. developed a novel algorithm for generation multi-level decision trees. This algorithm is based on the ant colony optimization model, in which a global ant population is divided into subpopulations. The calculation is performed simultaneously for each such subpopulation. The exchange of information between ants is possible through direct and indirect communication channels on the local and global (inter-subpopulation) levels. All ants cooperate with each other, and the whole system has a heterarchical structure. While Boryczka's methodology can be used as a prototype module of a realistic system, Otamendi presents the real-life implementation of a decision support simulation system in a car company, which is used in the process of redesigning of assembly cells.

We believe that all of the papers presented in this special issue will serve as a reference for students, researchers, and industry practitioners interested or currently working in the evolving and interdisciplinary area of scalable computing and intelligent networking. We hope that the readers will find new inspiration for their research.

We are grateful to all the contributors of this issue. We thank the authors for their time and efforts in the presentation of their recent research results. We would also like to express our sincere thanks to the reviewers, who have helped us to ensure the quality of this publication. Our special thanks go to Ngoc Thanh Nguyen (Editor-in-Chief) and the LNCS editorial staff of Springer with Alfred Hofmann, for supporting the TCCI journal and this publication.

January 2013

Joanna Kołodziej
Marenglen Biba
Tadeusz Burczyński

Transactions on Computational Collective Intelligence

This Springer journal focuses on research in computer-based methods of computational collective intelligence (CCI) and their applications in a wide range of fields such as the Semantic Web, social networks, and multi-agent systems. It aims to provide a forum for the presentation of scientific research and technological achievements accomplished by the international community.

The topics addressed by this journal include all solutions to real-life problems for which it is necessary to use CCI technologies to achieve effective results. The emphasis of the papers published is on novel and original research and technological advancements. Special features on specific topics are welcome.

Table of Contents

Markov Chain Based Analysis of Agent-Based Immunological System

Aleksander Byrski[1], Robert Schaefer[1], and Maciej Smołka[2]

[1] Dept. of Computer Science, AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
{olekb,schaefer}@agh.edu.pl
[2] Institute of Computer Science, Jagiellonian University,
ul. Łojasiewicza 6, 30-348 Kraków, Poland
smolka@ii.uj.edu.pl

Abstract. In the course of the paper we recall the Markov model for immunological Evolutionary Multi-Agent System. The model allows to study dynamic features of the computation and increases understanding the considered classes of systems. The main contribution of the paper is the draft of the proof of the ergodicity feature of the Markov chain modelling iEMAS dynamics.

1 Introduction

Certain heuristic system may never become one ultimate answer to solving all possible optimisation problem [22]. On the other hand, when building complex hybrid algorithms, an important question should be posed, does this system is able to work at all? This is important, because complex search methods may affect the ability to find all possible answers to the given problem, therefore formal proving of certain features of the computation becomes an important argument in the discussion of applicability of certain search methods.

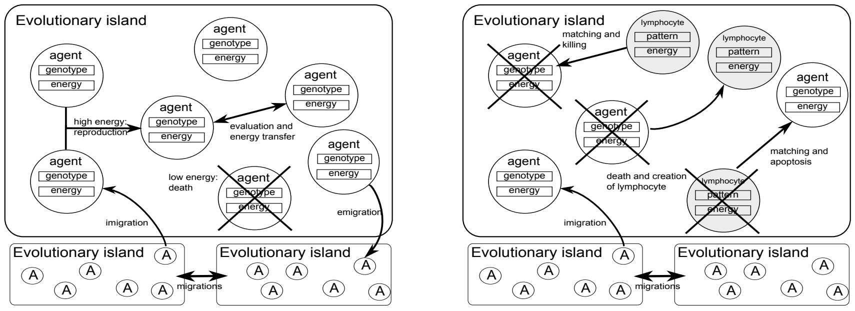
The formal model presented by Vose [20] proved in the most simple, yet effective way, asymptotic guarantee of success, i.e. "ability to find all local maximizers (minimizers) with probability 1 after infinite number of epochs" [18,12,15] in the analysis of the Simple Genetic Algorithm (SGA) behaviour, formally confirming the possibility of using SGA for global optimisation. Formal models for genetic algorithms were also proposed by other researchers, providing a deeper insight into the long term, steady state behaviour of large population EAs [9,19,16] or modelling specific features of EAs such as selection, genetic drift, niching etc. [10,14,11]. Many other, more complex, biologically-inspired computational techniques were proposed (e.g. memetic systems, immune-inspired systems), however, the problem of construction of appropriate mathematical models and approaches at proving asymptotic guarantee of success do not seem to be studied extensively or even were not undertaken at all.

In the course of paper we recall the basic features of the stochastic Markov models already introduced in the works of Byrski et al. (e.g. [5,17,6]). We define the space of states, synchronisation mechanism, and we draw the probability transition function. The main contribution of the paper is a draft of the sequence of actions proving the ergodicity of Markov chain constructed for iEMAS (immunological Evolutionary Multi Agent System introduced by Byrski and Kisiel-Dorohinicki [2]) by transferring the system between two arbitrarily chosen states. The formal proof for EMAS (predecessor of iEMAS, being a general optimisation system leveraging paradigms of evolutionary computation and agency, introduced by Cetnarowicz [7]) ergodicity has already been submitted for publication, we will follow with full proof of ergodicity of iEMAS in the near future.

2 Evolutionary and Immunological Agent-Based Computation

EMAS and iEMAS are general-purpose optimisation systems leveraging paradigms of evolutionary computation and agency, following work of Cetnarowicz [7]) that has already proven its efficiency for certain class of problems (see e.g., [4,2,3]).

In the simplest possible model of an evolutionary multi-agent system there is one type of agents and one resource defined. Genotypes of agents represent feasible solutions to the problem.



(a) EMAS (b) iEMAS

Fig. 1. Evolutionary (EMAS) and immunological (iEMAS) multi-agent system

Energy is exchanged by agents in the process of evaluation. The agent increases its energy when it finds out that one (e.g. randomly chosen) of its neighbours, has lower fitness. In this case, the agent takes part of its neighbour's energy, otherwise, it passes part of its own energy to the evaluated neighbour. The level of life energy triggers actions of death and reproduction (low energy causes death while high energy makes reproduction possible). Attaining predefined level of energy may lead an agent also to migrate from one evolutionary island to another (see Fig. 1(a)).

Immune-inspired approaches were applied to many problems, such as classification or optimisation (e.g. [8]). The most often used algorithms of clonal and negative selection correspond to their origin and are used in a variety of applications [21].

The main idea of applying immunological inspirations to speed up the process of selection in EMAS is based on the assumption that 'bad' phenotypes come from 'bad' genotypes. Thus, a new group of agents (acting as lymphocyte T-cells) may be introduced [3]. They are responsible for recognising and removing agents with genotypes similar to the genotype pattern possessed by these lymphocytes. Another approach may introduce specific penalty applied by T-cells for recognised agents (certain amount of the agent's energy is removed) instead of removing them from the system. The general structure of iEMAS (immunological EMAS) is presented in Fig. 1(b).

Of course there must exist some predefined affinity (lymphocyte-agent matching) function, which may be based, e.g., on the percentage difference between corresponding genes. Agents-lymphocytes are created in the system after the action of death. The late agent genotype is transformed into lymphocyte patterns by means of mutation operator, and the newly created lymphocyte (or group of lymphocytes) is introduced into the system.

In both cases, new lymphocytes must undergo the process of negative selection. In a specific period of time, the affinity of immature lymphocytes' patterns to 'good' agents (possessing relatively high amount of energy) is tested. If it is high (lymphocytes recognize 'good' agents as 'non-self') they are removed from the system. If the affinity is low, it is assumed that they will be able to recognize 'non-self' individuals ('bad' agents) leaving agents with high energy intact. The life span of lymphocytes is controlled by specific, renewable resource (strength), used as a counter by the lymphocyte agent (see Fig. 1(b)).

3 Agent-Based Management and Synchronisation

We start considerations from evolutionary multi-agent systems solving global optimisation problems (cf. [17]), which consist in finding all global minimizers of a given nonnegative fitness function over a finite genetic universum U with cardinality r. EMAS agents belong to a predefined finite set Ag. Every active agent is assigned to a location (evolutionary island) from the set $Loc = \{1, \ldots, s\}$. The locations are interconnected with channels along which agents can migrate. A channel topology is given by a symmetric relation $Top \subset Loc^2$.

Continuing considerations presented in e.g. [5,17] we focus on the *Immunologically based Evolutionary Multi-Agent System* (iEMAS) that contains (besides dynamic collection of agents that belong to the predefined finite set Ag identical to the one of EMAS) a dynamic collection of lymphocytes that belong to the finite set Tc. Lymphocytes are unambiguously indexed by the genotypes from U, so that $\#Tc = \#U = $ r.

The lymphocytes have a similar structure as the agents previously defined, however, their actions differ (because their goals differ from the agents' goals) and their total energy does not have to be constant.

iEMAS may be modeled as the following tuple:

$$< U, \{P_i\}_{i \in Loc}, Loc, Top, Ag, \{agsel_i\}_{i \in Loc}, locsel, \{LA_i\}_{i \in Loc}, MA, \omega, Act, \\ \{typesel_i\}_{i \in Loc}, \{tcsel_i\}_{i \in Loc}, Tc, Tcact > \qquad (1)$$

where:

- MA (master agent) is used to synchronize the work of the locations; it allows to perform actions in particular locations. This agent is also used to introduce necessary synchronisation into the system.
- $locsel : X \to \mathcal{M}(Loc)$ is the function used by MA to determine which location should be allowed to perform the next action,
- LA_i (local agent) is assigned to each location; it is used to synchronize the work of computational agents present in its location, LA_i chooses the computational agent and lets it evaluate a decision and perform the action, at the same time asking MA whether this action may be performed.
- $agsel_i : X \to \mathcal{M}(U \times P_i)$ is a family of functions used by local agents to select the agent that may perform the action, so every location $i \in Loc$ has its own function $agsel_i$.
- $\omega : X \times U \to \mathcal{M}(Act)$ is a function used by agents for selecting actions from the set Act; both these symbols will be described later.
- Act is a predefined, finite set of actions.
- $typesel_i$ is a function used to select the type of agent in i-th location to interact with the system in the current step,
- $tcsel_i$ is used to choose a lymphocyte in i-th location to interact with the system in the current step,
- φ is the decision function for lymphocytes,
- $Tcact$ is a set of actions that may be performed by lymphocytes.

Hereafter $\mathcal{M}(\Omega)$ shall stand for the space of probabilistic measures over Ω.

In order to design a Markov model of the system with relaxed synchronisation (i.e. such that agents present in different locations may act in parallel), a timing mechanism must be introduced, i.e. all state changes must be assigned to subsequent time moments $t_0, t_1, \ldots$.

In Fig. 2 the scheme of the synchronisation mechanism built using agents, $LA_i, i \in Loc$ and MA is presented.

The computational agent CA present in the location i in every observable time moment chooses an action it wants to perform and asks its supervisor (local agent LA_i) for a permission to carry on. Then it suspends its work waiting for the permission. When the permission is granted and the decision assigned to the considered action is positive, the computational agent changes the state of the location. Afterwards the agent suspends its work again in order to get a permission to perform a subsequent action. The immunological agent TC works in a similar way to CA, managing behaviour of a single lymphocyte.

The local agent LA_i receives signals containing actions to be performed from all its agents. Then chooses one computational agent which should try to perform its action. This action is reported to the master agent MA and after receiving

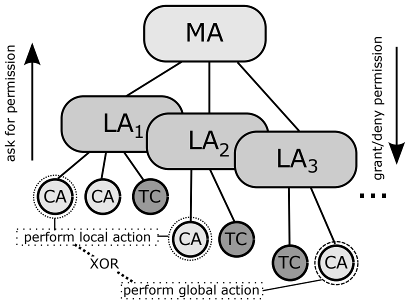


Fig. 2. Scheme of the synchronisation mechanism

permission, the computational agent can perform the action. All other agents are stopped from performing their actions.

The master agent MA waits for all requests from location and then chooses randomly one location. If this location asks for permission to perform global action, then it is granted this permission and all other locations are rejected. Otherwise all locations which asked for the permission to perform global action are rejected and all those asking for permission to perform local action — are granted.

4 System State

In this section we cite the description of EMAS state and extend it by adding a matrix describing iEMAS state (following [17]).

4.1 EMAS State

Let us introduce the set of three-dimensional, incidence and energy matrices $x \in X$ with s layers (corresponding to all locations) $x(i) = \{x(i, gen, n),\ gen \in U,\ n \in P_i\},\ i \in Loc$. The layer $x(i)$ will contain energies of agents in i-th location. In other words, if $x(i, gen, k) > 0$, it means that the k-th clone of the agent containing the gene $gen \in U$ is active, its energy equals $x(i, gen, k)$ and it is located in i-th location.

We introduce the following coherency conditions:

- $(\cdot, j, k)$-th column contains at most one value greater than zero, which expresses that the agent with k-th copy of j-th genotype may be present in only one location at a time, whereas other agents containing copies of j-th genotype may be present in other locations;
- incidence and energy matrices' entries are non-negative $x(i, j, k) \geq 0,\ \forall\, i = 1, \ldots, s,\ \ j = 1, \ldots, r,\ \ k = 1, \ldots, p$ and $\sum_{i=1}^{s} \sum_{j=1}^{r} \sum_{k=1}^{p} x(i, j, k) = 1$, which means that total energy contained in the whole system is constant, equal to 1;

- each layer $x(i)$ contains at most q_i values greater than zero, which denotes the maximum capacity of the i-th location, moreover, the quantum of energy Δe is lower or equal than total energy divided by the maximal number of individuals that may be present in the system $\Delta e \leq \frac{1}{\sum_{i=1}^{s} q_i}$ which allows us to achieve maximal population of agents in the system;
- reasonable values of p should be greater or equal to 1 and less or equal to $\sum_{i=1}^{s} q_i$; we assume that $p = \sum_{i=1}^{s} q_i$ which assures that each configuration of agents in locations is available, respecting the constrained total number of active agents $\sum_{i=1}^{s} q_i$; increasing p over this value does not enhance the descriptive power of the presented model;
- the maximal number of copies for each location $\#P_i$ should not be less than q_i, because we want to allow a system state in which a particular location is filled with clones of one agent; obviously increasing $\#P_i$ over q_i is only a formal constraint relaxation, so finally we assume that $\#P_i = q_i$.

Gathering all these conditions, the set of three-dimensional incidence and energy matrices may be described in the following way.

$$\begin{aligned} \Lambda = \Big\{ & ince \in \{0, \Delta e, 2 \cdot \Delta e, 3 \cdot \Delta e, \ldots, m \cdot \Delta e\}^{s \cdot r \cdot p}, \Delta e \cdot m = 1, \\ & \sum_{i=1}^{s}\sum_{j=1}^{r}\sum_{k=1}^{p} x(i,j,k) = 1, \forall\ i = 1, \ldots, s: \ \sum_{j=1}^{r}\sum_{k=1}^{p}[x(i,j,k) > 0] \leq q_i, \\ & \forall\ i = 1, \ldots, s,\ j = 1, \ldots, r,\ k \notin P_i:\ x(i,j,k) = 0, \\ & \forall j = 1, \ldots, r,\ k = 1, \ldots, p: \ \sum_{i=1}^{s}[x(i,j,k) > 0] \leq 1 \Big\} \end{aligned} \tag{2}$$

where $[\cdot]$ denotes the value of the logical expression contained in the parentheses.

4.2 iEMAS State

In addition to the EMAS state describing the location and energy of agents (see (4.1)), we need to consider a set of matrices containing similar information for lymphocytes. Yet there is no need to assure the constant total energy for lymphocytes. We describe this additional set of lymphocyte incidence and energy matrices in the following way:

$$\begin{aligned} \Gamma = \Big\{ & tcince \in [0, \Delta e, \ldots, n \cdot \Delta e]^{r \cdot s} : \forall\ i = 1, \ldots, s \ \ \sum_{j=1}^{r}[tcince(i,j) > 0] \leq tcq_j \\ & \text{and } \forall\ j = 1, \ldots, r \ \ \sum_{i=1}^{s}[tcince(i,j) > 0] \leq 1 \Big\} \end{aligned} \tag{3}$$

where $tcince(i,j)$ stands for energy of tc_j being active in the location i. The integers tcq_j, $j = 1, \ldots, s$ stand for the maximum number of lymphocytes in particular locations. It is most convenient to assume $tcq_j = q_j$, $\forall j = 1, \ldots, s$.

The space of iEMAS states is defined as follows:

$$X = \Lambda \times \Gamma \tag{4}$$

5 System Behaviour

Let us denote by X_{gen} the subset of states in which there are active agents with genotype $gen \in U$ or an active lymphocyte.

5.1 EMAS Behaviour

Each action $\alpha \in Act$ will be represented as the pair of function families $(\{\delta_\alpha^{gen}\}_{gen \in U}, \{\vartheta_\alpha^{gen}\}_{gen \in U})$. The functions

$$\delta_\alpha^{gen} : X \to \mathcal{M}(\{0,1\}) \tag{5}$$

represent the decision to be taken: whether the action can be performed or not. The action α is performed with the probability $\delta_\alpha^{gen}(x)(1)$ by the agent $ag_{gen,n}$ at the state $x \in X$ and rejected with the probability $\delta_\alpha^{gen}(x)(0)$.

Next, the formula

$$\vartheta_\alpha^{gen} : X \to \mathcal{M}(X) \tag{6}$$

defines the non-deterministic state transition functions, so that ϑ_α^{gen} is caused by the execution of the action α by the agent $ag_{gen,n}$. Because the function is invoked only if the agent in active, it is enough to define its restriction $\vartheta_\alpha^{gen}|X_{gen}$ and take an arbitrary value on $X \setminus X_{gen}$.

If any action is rejected, the trivial state transition

$$\vartheta_{null} : X \to \mathcal{M}(X) \tag{7}$$

such that for all $x \in X$

$$\vartheta_{null}(x)(x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

is performed.

The probability transition function for the action α performed by the agent containing the genotype *gen*

$$\varrho_\alpha^{gen} : X \to \mathcal{M}(X) \tag{9}$$

is given by the formula

$$\begin{aligned} \varrho_\alpha^{gen}(x)(x') = \delta_\alpha^{gen}(x)(0) \cdot \vartheta_{null}(x)(x') \\ + \delta_\alpha^{gen}(x)(1) \cdot \vartheta_\alpha^{gen}(x)(x') \end{aligned} \tag{10}$$

where $x \in X$ denotes a current state and $x' \in X$ a consecutive state resulted from the conditional execution of α.

5.2 iEMAS Behaviour

We introduce the function $typesel_i$ choosing which type of agents will have the possibility of performing the action:

$$typesel_i : X \to \mathcal{M}(\{0,1\}) \tag{11}$$

when 0 is chosen, one of the agents is activated, when 1 — the lymphocyte.

The function choosing which agent will be activated $agsel_i$ is like in EMAS but it now depends in some way on the extended state from X defined by (4). Now we introduce a new function that will choose which lymphocyte will be activated:

$$tcsel_i : X \to \mathcal{M}(Tc) \tag{12}$$

The function ω choosing the action for the active agent remains intact, though its domain changes (because of the new state definition, see (4)).

The function choosing the action for the active lymphocyte is the following:

$$\varphi : U \times X \to \mathcal{M}(Tcact) \tag{13}$$

We will use the family of functions $\eta_\alpha^{gen} : X \to \mathcal{M}(X)$ where $gen \in U$, $\alpha \in Tcact$. Each of them expresses the probability transition imposed by the lymphocyte tc_{gen} that performs the action $\alpha \in Tcact$. They are given by the general formula:

$$\eta_\alpha^{gen}(x)(x') = \gamma_\alpha(gen, x)(\{0\}) \cdot \vartheta_{null}(x)(x') + \gamma_\alpha(gen, x)(\{1\}) \cdot \kappa_\alpha^{gen,n}(x)(x') \tag{14}$$

The agents' and lymphocytes' actions may be divided into two distinct types: global — they change the state of the system in two or more locations, so only one global action may be performed at a time, local — they change the state of the system inside one location respecting only the state of local agents, only one local action for one location may be performed at a time.

Therefore we divide the Act set in the following way: $Act = Act_{gl} \cup Act_{loc}$ and accordingly, $Tcact : Tcact = Tcact_{gl} \cup Tcact_{loc}$. Speaking informally, local actions (elements of $Act_{loc}, Tcact_{loc}$) change only the entries of the layer $x(i)$ of the incidence and energy matrices if the location $i \in Loc$ contains the agent performing a certain action. Moreover, these actions do not depend on other layers of x. The action *null* is obviously "the most local one", because it does not change anything at all.

In the case of EMAS and iEMAS, actions such as evaluation or lymphocyte pattern matching may be perceived as local, at the same time action of migration is perceived as global. The above-stated conditions may be defined formally and may be used to prove commutativity of iEMAS (cf. [5,17]), however here we skip this proof because lack of space.

6 Parallel iEMAS Dynamics

At the observable moment at which EMAS takes the state $x \in X$ all agents in all locations notify their local agents their intent to perform an action, all local agents choose an agent with the distribution given by the $agsel_i(x)$, $i \in Loc$ function and then notify the master agent of their intent to let perform an action by one of their agents. The master agent chooses the location with the probability distribution given by $locsel(x)$.

We extend the model of EMAS dynamics in order to model the behaviour of iEMAS. The probability that in the chosen location $i \in Loc$ the agent or lymphocyte wants to perform local action is as follows:

$$\xi_i(x) = typesel(x)(\{0\}) \sum_{gen \in U} \sum_{n=1}^{p} (agsel_i(x)(\{gen, n\}) \\ \cdot \omega(gen, x)(Act_{loc})) + typesel(x)(\{1\}) \tag{15}$$

The probability that the master agent will chose the location with the agent intending to perform the local action is:

$$\zeta^{loc}(x) = \sum_{i \in Loc} locsel(x)(\{i\}) \xi_i(x) \tag{16}$$

of course the probability of choosing the global action by the master agent is:

$$(1 - \zeta^{loc}(x)) = \zeta^{gl}(x) \tag{17}$$

If the global action is chosen, the state transition is given by:

$$\tau^{gl}(x)(x') = \sum_{i \in Loc} locsel(x)(\{i\}) \cdot \left(\sum_{gen \in U} \sum_{n=1}^{p} agsel(x)(\{gen, n\}) \cdot \\ \left(\sum_{\alpha \in Act_{gl}} \omega(gen, x)(\{\alpha\}) \cdot \varrho_{\alpha}^{gen,n}(x)(x') \right) \right) \tag{18}$$

Let us state the set of action sequences containing at least one local action:

$$Act_{+1loc} = \Bigg\{ (\alpha_1, \ldots, \alpha_s) \in (Act \cup Tcact)^s; \\ \sum_{i=1}^{s} [\alpha_i \in (Act_{loc} \cup Tcact)] > 0 \Bigg\} \tag{19}$$

The probability that in i-th location the agent ag_{gen_i,n_i} or the lymphocyte $tc_{\widetilde{gen_i}}$ chooses the action α_i is given by:

$$\mu_{\alpha_i,gen_i,n_i,\widetilde{gen_i}}(x) = typesel(x)(\{0\}) \cdot agsel_i(x)(\{gen_i, n_i\}) \omega(gen_i, x)(\{\alpha_i\}) + \\ typesel(x)(\{1\}) tcsel_i(x)(\{\widetilde{gen_i}\}) \varphi(\widetilde{gen_i}, x)(\{\alpha_i\}) \tag{20}$$

Let us define a multi-index:

$$ind = (\alpha_1, \ldots, \alpha_s; (gen_1, n_1), \ldots, (gen_s, n_s);\ (\widetilde{gen_1}), \ldots, (\widetilde{gen_s})) \\ \in IND = (Act \cup Tcact)^s \times (U \times \{1, \ldots, p\})^s \times U^s \tag{21}$$

the probability that in consecutive locations agents ag_{gen_i,n_i} or lymphocytes $tc_{\widetilde{gen_i}}$ will choose the actions α_i is given by:

$$\mu_{ind}(x) = \prod_{i=1}^{s} \mu_{\alpha_i,gen_i,n_i,\widetilde{gen_i}}(x). \tag{22}$$

Transition function for parallel system is following:

$$\tau^{loc}(x)(x') = \sum_{(\alpha_1,\ldots,\alpha_s)\in Act_{+1loc}} \sum_{ind\in IND} \mu_{ind}(x)(\pi_1^{ind}(x)\circ, \ldots, \circ\, \pi_s^{ind}(x))(x') \tag{23}$$

where π_i is defined as:

$$\pi_i^{ind}(x) = \begin{cases} \varrho_{\alpha_i}^{gen_i,n_i}(x), & \alpha_i \in Act_{loc} \\ \eta_{\alpha_i}^{\widetilde{gen_i}}(x), & \alpha_i \in Tcact \\ \vartheta_{null}, & \alpha_i \in Act_{gl} \end{cases} \tag{24}$$

The value of $(\pi_1^{ind}(x)\circ, \ldots, \circ\, \pi_s^{ind}(x))(x')$ does not depend on the composition order, because transition functions associated with local actions commutate pairwise (see 5.2) . Finally, we may derive the following observation.

Observation 1. *The probability transition function for the parallel iEMAS model is given by the formula*

$$\tau(x)(x') = \zeta^{gl}(x)\tau^{gl}(x)(x') + \zeta^{loc}(x)\tau^{loc}(x)(x') \tag{25}$$

and formulas (15) – (24).

Observation 2. *The stochastic state transition of iEMAS given by formula (25) satisfies the Markov condition.*

Proof. All transition functions and probability distributions given by formulas (15)–(24) depend only on the current state of the system, which motivates the Markovian features of the transition function τ given by (25). The transition functions do not depend on the number of step at which is applied what motivates the stationarity of the chain.

7 iEMAS Ergodicity Proof Draft

In this section we present a draft of a proof of the ergodicity feature for the Markov chain describing the behaviour of iEMAS.

Theorem 1. *Assume that the following assumptions hold.*

1. *The migration energy threshold is lower than the total energy divided by the number of locations $e_{migr} < \frac{1}{s}$. This assumption ensures that there will be at least one location in the system in which an agent is capable of performing migration (by gathering enough energy from its neighbors).*

2. *The quantum of energy is lower than or equal to the total energy divided by the maximum number of agents that may be present in the system* $\Delta e \leqslant \frac{1}{\sum_{i=1}^{s} q_i}$ *. This assumption allows to achieve a maximal population of agents in the system.*
3. *Reproduction (cloning) energy is lower than two energy quanta* $e_{repr} \leqslant 2\Delta e$*.*
4. *The amount of energy passed from parent to the child during cloning action is equal to* Δe *(so* $n_1 = 1$*).*
5. *The maximum number of agents on every location is greater than one,* $q_i > 1, i = 1, \ldots, s$*.*
6. *Locations are totally connected, i.e.* $Top = Loc^2$*.*
7. *Each active agent can be selected by its local agent with strictly positive probability.*
8. *The families of probability distributions being the parameters of EMAS have uniform, strictly positive lower bounds.*

Then the Markov chain modeling iEMAS (see equation (25)) is irreducible, i.e. all its states communicate.

In order to prove the Theorem 1, it is enough to show that the passage from x_b to x_e (two arbitrarily chosen states from X) may be performed in a finite number of steps with probability strictly greater then zero.

Let us consider the following sequence of stages.

- **Stage 0:** In every location in parallel: If the location is full, an agent is chosen, and it performs sequentially evaluation action with one of its neighbors in order to remove it (to make possible incoming migration from any other location, in case this location is full). After removing one of its neighbors the agent tries to perform any global action, e.g., migration (and fails), until the end of the stage. Otherwise, the trivial null state transition is performed. Final state of the Stage 0 is denoted by x_{0e}.
- **Stage 1 a:** One location is chosen, at which the sum of agents' energy exceeds the migration threshold in the state x_{0e} (based on assumption 1 of Theorem 1 there must be at least one). Then one agent from this location ag_{gen_1,n_1} (possibly possessing the largest energy in the state x_{0e}) is chosen. This agent performs a sequence of evaluation actions in order to gather all energy from all its neighbors, finally removing them from the system (by bringing their energy to zero).
- **Stage 1 b:** If there are any lymphocytes on the current location, they perform killing action, one by one, on the agent ag_{gen_1,n_1}, failing to remove it from the system, until all lymphocytes are removed. In the end, only one agent is present in the location.
- **Stage 1 c:** Now this agent begins the first migration round in order to visit all locations and to remove the agents (overtaking their energy by performing multiple *get* actions) and remove all lymphocytes. This round is finished at location i_1. Now, agent ag_{gen_1,n_1} possesses the total energy of the system which equals 1. Final state of the Stage 1 is denoted by x_{1e}. Note, that the state matrix has only one positive entry $x_{1e}(i_1, gen_1, n_1) = 1$.

- **Stage 2 a:** The agent performs cloning action producing one of the agents (ag_{gen_2,n_2}) that will be present on the location i_2, one of the locations in the state x_e containing total energy greater than the migration threshold. Now it passes all of its energy to this newly produced agent, finally being removed from the system. The purpose of the Stage 2 is to ensure that the agent recreating the population at the last location i_2 will be one of the agents present on this location in the state x_e. Otherwise if i_2 is full in the state x_e, ag_{gen_1,n_1} could not recreate this population. If ag_{gen_1,n_1} is active at the location i_2 at the state x_e (i.e. $x_e(i_2, gen_1, n_1) > 0$), the Stage 2 may be omitted (in this case ag_{gen_1,n_1} takes the role of ag_{gen_2,n_2} in the following stages).
- **Stage 3:** Next, the agent ag_{gen_2,n_2} begins the second migration round (starting migration from the location i_1) visiting all locations. In every visited location it performs cloning action producing one of the agents that will be present on this location in the state x_e. The cloned agent on each non-empty location (denoted by $ag_{gen_i^{first},n_i^{first}}$) will receive the total energy that should be assigned to its location, by the sequence of evaluation actions. The agent finishes the migration after recreating the population on the location i_2 (one of the islands containing a total energy in the state x_e greater than the migration threshold).
- **Stage 4 a:** In the system, the following sequence of actions assigned with the consecutive locations labeled by $i \in Loc$, non empty in the state x_e, is performed: every agent $ag_{gen_i^{first},n_i^{first}}$ perform a cloning action to produce an agent with the genotype of one of lymphocytes existing in the location in the state x_e. Now it performs a sequence of evaluation action to remove the agent (and the appropriate lymphocyte is performed). The lymphocyte performs a sequence of energy lowering actions to adjust its energy to the level observed in the state x_e. This is repeated until all the lymphocytes present in x_e are recreated.
- **Stage 4 b:** In the system, the following sequence of actions assigned with the consecutive locations labeled by $i \in Loc$, non empty in the state x_e, is performed: every agent $ag_{gen_i^{first},n_i^{first}}$ performs a sequence of cloning actions, recreating the population of agents on its location in the state x_e.
- **Stage 5:** In every location in parallel: agent $ag_{gen_i^{first},n_i^{first}}$ performs a sequence of evaluation actions with its neighbors in order to pass to them a sufficient amount of energy, required in the state x_e.

In the extended version of this paper we will show that every of aforementioned stages requires performing at most finite number of Markov chain steps by estimating their upper bounds. Moreover, we will show, that every aforementioned sequences have non-zero probabilities by estimating its lower bounds.

Theorem 1 leads us straightforwardly to the statement that every possible state of iEMAS is reachable (with positive probability) after performing a finite sequence of transitions independently on the initial population. We can reformulate such a conclusion in the following corollary.

Corollary 1. *All states containing the extrema are reachable from an arbitrary initial state. Thus iEMAS satisfies asymptotic guarantee of success in the sense of [18,12,15].*

The following theorem shows an additional feature of the considered Markov chain.

Theorem 2. *If the assumptions of Theorem 1 hold, then the Markov chain modeling EMAS is aperiodic.*

Proof. Let us consider a state of the chain such that every location contains a single computational agent only. In this case let us assume that each agent chooses evaluation as its next action. Because all agents have chosen local actions, the master agent will allow them all to perform their actions, however the absence of neighbors will force all the agents to perform the trivial (i.e. null) action. The transition probability function is then the s-fold composition of ϑ_{null}. Therefore in this case the system will return to the same state in one step. The probability of such transition is greater than zero. It means that the considered state is aperiodic. Our chain is irreducible (see Theorem 1) and therefore it has only one class of states, the whole state space, which obviously contains the considered aperiodic state. On the other hand, from Theorem 2.2 of [13] we know that aperiodicity is a state class property. In our case it means that all states of EMAS are aperiodic, which concludes the proof.

The following corollary is a consequence of Theorems 1 and 2.

Corollary 2. *The Markov chain modeling EMAS is* ergodic.

Remark 1. It is worth noticing that the Markov chain (25) is ergodic in a strong sense (not only irreducible, but also aperiodic). Such chains are quite often called *regular* (see e.g. [13]).

Because the space of states X is finite we may introduce the probability transition matrix:

$$Q = \{\tau(x)(y)\},\ x, y \in X \tag{26}$$

where τ is the iEMAS probability transition function — see Eq. (25). The Markov chain describing the iEMAS dynamics is a sequence of random variables (or, equivalently, probability distributions) $\{\xi_t\} \subset \mathcal{M}(X), t = 0, 1, \ldots$ where ξ_0 should be a given initial probability distribution. Of course we have that

$$\xi_{t+1} = Q \cdot \xi_t,\ t = 0, 1, \ldots \tag{27}$$

Remark 2. From Theorems 1 and 2 as well as the ergodic theorem [1] there exists a strictly positive limit $\widehat{\xi} \in \mathcal{M}(X)$ (i.e., $\widehat{\xi}(x) > 0, \forall\ x \in X$) of the sequence $\{\xi_t\}$ as $t \to +\infty$. This equilibrium distribution does not depend on the initial probability distribution ξ_0.

8 Conclusions

In the course of this contribution, a formal model for iEMAS has been recalled and adjusted for a discrete system state space. The space of states and the transition functions allowing to construct a uniform Markov chain model have been proposed. This model based on stationary Markov chains allows a better understanding of the behaviour of the proposed complex systems as well as their constraints.

One of the main implications of the analysis conducted here is the formulation and proof draft of Theorem 1 stating that the Markov chain based model of EMAS is stationary and ergodic. This will lead to an important conclusion stated in Corollary 1, namely that EMAS possesses the feature of asymptotic guarantee of success.

Ergodicity of Markov chain modelling iEMAS proves that this hybridization does not hamper the capabilities of solving optimization problem in general, the experimental results prove, that for certain problems with complex fitness function (e.g., evolution of neural network parameters) employing iEMAS is especially advantegeous.

A full formal proof for EMAS ergodicity has already been formulated and submitted for publication. In the near future we will follow with preparing a similar proof of iEMAS ergodicity.

References

1. Billingsley, P.: Probability and Measure. Wiley-Interscience (1995)
2. Byrski, A., Kisiel-Dorohinicki, M.: Immunological selection mechanism in agent-based evolutionary computation. In: Proc. of IIS: IIPWM 2005 Conference, Gdansk, Poland. Advances in Soft Computing. Springer (2005)
3. Byrski, A., Kisiel-Dorohinicki, M.: Agent-based evolutionary and immunological optimization. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007, Part II. LNCS, vol. 4488, pp. 928–935. Springer, Heidelberg (2007)
4. Byrski, A., Kisiel-Dorohinicki, M., Nawarecki, E.: Agent-Based Evolution of Neural Network Architecture. In: Hamza, M. (ed.) Proc. of the IASTED Int. Symp.: Applied Informatics, IASTED/ACTA Press (2002)
5. Byrski, A., Schaefer, R.: Stochastic model of evolutionary and immunological multi-agent systems: Mutually exclusive actions. Fundamenta Informaticae 95(2-3), 263–285 (2009)
6. Byrski, A., Schaefer, R., Smołka, M.: Asymptotic features of parallel agent-based immunological system. In: Burczyński, T., Kołodziej, J., Byrski, A., Carvalho, M. (eds.) Proc. of 25th European Conference on Modelling and Simulation (2011)
7. Cetnarowicz, K., Kisiel-Dorohinicki, M., Nawarecki, E.: The application of evolution process in multi-agent world (MAW) to the prediction system. In: Tokoro, M. (ed.) Proc. of the 2nd Int. Conf. on Multi-Agent Systems (ICMAS 1996). AAAI Press (1996)
8. Dasgupta, D., Nino, L.: Immunological Computation Theory and Applications. Auerbach (2008)

9. Davis, T.E., Principe, J.C.: A simulated annealing like convergence theory for the simple genetic algorithm. In: Proc. of the Fourth International Conference on Genetic Algorithms, San Diego, CA, pp. 174–181 (1991)
10. Goldberg, D., Segrest, P.: Finite Markov chain analysis of genetic algorithms. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and their Application, pp. 1–8. L. Erlbaum Associates Inc., Hillsdale (1987)
11. Horn, J.: Finite Markov Chain Analysis of Genetic Algorithms with Niching. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 110–117. Morgan Kaufmann (1993)
12. Horst, R., Pardalos, P.: Handbook of Global Optimization. Kluwer (1995)
13. Iosifescu, M.: Finite Markov Processes and Their Applications. John Wiley and Sons (1980)
14. Mahfoud, S.: Finite Markov Chain Models of an Alternative Selection Strategy for the Genetic Algorithm. Complex Systems 7, 155–170 (1991)
15. Rinnoy Kan, A., Timmer, G.: Stochastic global optimization methods. Mathematical Programming 39, 27–56 (1987)
16. Rudolph, G.: Massively parallel simulated annealing and its relation to evolutionary algorithms. Evolutionary Computation 1, 361–383 (1994)
17. Schaefer, R., Byrski, A., Smołka, M.: Stochastic model of evolutionary and immunological multi-agent systems: Parallel execution of local actions. Fundamenta Informaticae 95(2-3), 325–348 (2009)
18. Schaefer, R.: Foundations of global genetic optimization. Springer (2007)
19. Suzuki, J.: A Markov Chain Analysis on a Genetic Algorithm. In: Forrest, S. (ed.) Proc. of the 5th ICGA, pp. 146–154. Morgan Kaufmann (1993)
20. Vose, M.: The Simple Genetic Algorithm: Foundations and Theory. MIT Press, Cambridge (1998)
21. Wierzchoń, S.: Function optimization by the immune metaphor. Task Quaterly 6(3), 1–16 (2002)
22. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)

Towards Dynamic Orchestration of Semantic Web Services

Domenico Redavid[1], Stefano Ferilli[2], and Floriana Esposito[2]

[1] Artificial Brain Srl - Bari, Italy
redavid@abrain.it
http://www.abrain.it

[2] Dipartimento di Informatica, Università degli Studi di Bari, Italy
{ferilli,esposito}@di.uniba.it
http://www.di.uniba.it

Abstract. The Semantic Web together with Web services technologies enable new scenarios in which the machines use the Web to provide intelligent services in an autonomus way. The orchestration of Semantic Web Services now can be defined from an abstract perspective where their formal semantics can be exploited by software agents to replace human input. This paper tackles the more difficult use case, automatic composition, providing a complete solution to create and manage service processes in a semantically interoperable environment.

Keywords: Semantic Web Services, Orchestration, Process generation.

1 Introduction

From the Web services perspective, an orchestration is a declarative specification that describes a work-flow supporting the execution of a specific business process, operation, or service [12]. Currently, Web services technologies allow to describe orchestration, but only at design-time. Semantic Web Services (SWSs) provide an ontological framework for describing services in a machine-readable format. In [10] a software agent is proposed that applies logical reasoning on SWS descriptions in order to provide on-the-fly orchestration capabilities. With the adjective *dynamic* we denote the capability of an agent to design and manage in an automatic way an orchestration schema using the semantic descriptions of available services. The notion of *dynamic* orchestration becomes useful in scenarios where there is the need of run-time designing process integration using semantic descriptions of the involved entities. This work addresses the most difficult part for the realization of this scenario, i.e., the automatic composition mechanism. In fact, as will explained in the following, orchestration is mainly a representation problem that requires the semantic representation of constraints (preconditions, effects, etc.), currently not properly supported by the Semantic Web standard languages.

This paper is organized in four sections. Section 2 describes what orchestration means from the Semantic Web perspective, the requirements to realize it and the support offered by the main SWS languages. Section 3 provides details about the language that better satisfies those requirements, to be used for the implementation of the composition use case. Section 4 describes the entire procedure to obtain a composed service written in the chosen language, while Section 5 proposes a complete example of the applied procedure. Finally, conclusions and future works complete the paper.

2 SWS Dynamic Orchestration

2.1 Orchestration and SWS Infrastructure

Orchestration for Web services must necessarily rely on XML-based descriptions. Their merely syntactical nature [12] represents an unsurmountable obstacle towards the automatic implementation of the operations that are necessary to accomplish dynamicity in orchestration. Let us, indeed, consider the following scenario:

> "Given a request (goal), an agent (dynamic orchestrator) *discovers* the possible SWSs able to accomplish the goal. At the same time, it *composes* the discovered services in order to have many possible ways to reach the goal. Having different alternatives available, it *selects* the best one exploring functional and non-functional properties of the different SWSs. The selection process can be used also during composition (sub-goal matching). Finally, the same agent manages the correct *invocation* of the selected services."

In this paper, dynamic orchestration of SWSs is inteded as carrying out one or more of the operations mentioned above. This is very different from the design of a work-flow for the execution of simple Web services [12]. The first step to realize this scenario is the automatization of the emphasized operations (discovery, composition, selection and invocation). These operations, amongst others (namely, publishing, deployment and ontology management) which are not directly involved in our scenario, were already identified as use cases for the SWS infrastructures into the *Usage Activities* dimension described in [3]. In the same work the architectural components (*Architecture* dimension) and semantic descriptions (*Service Ontologies* dimensions) needed to realize the

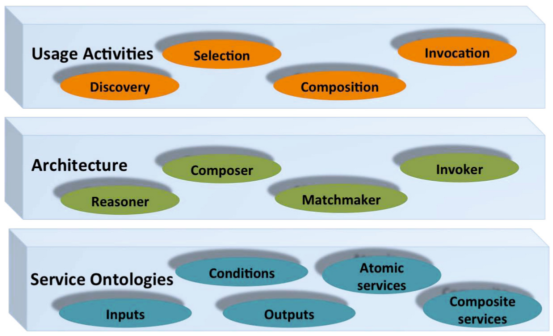


Fig. 1. Orchestration in the SWS Infrastructure

Usage Activities have been identified and used to define SWS infrastructures as formed by these three orthogonal dimensions. The *Architecture* includes a register, a reasoner, a matchmaker, a composer, and an invoker. The *Service Ontologies* dimension specifies the semantics of:

- Functional capabilities, such as inputs, output, and conditions (pre-conditions, effects, etc.);
- Non-Functional capabilities, such as category, cost and quality of service;
- Provider-related information, such as company name and address;
- Task or goal-related information and domain knowledge, defining, for instance, the type of the inputs of the service.

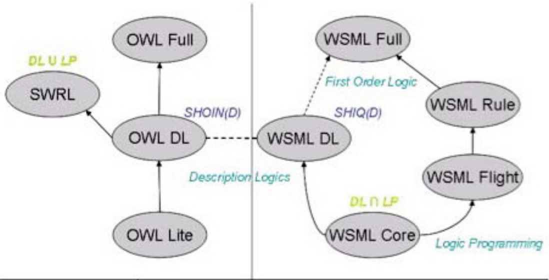
The dynamic SWS orchestration, as defined above, needs to be matched with these dimensions in order to identify its requirements. The results of this investigation are summarized in Figure 1. In the architecture dimension, the *reasoner* is the most important component to implement these use cases because it allows the semantic *matchmaking* of the service properties enabling the automatic *composition* of the services, and the automatic choice of the suitable parameters for the *invocation* of the services. The service ontologies dimension is necessary because it allows to define the semantics of functional and non-functional properties owned by atomic and composite services that participate to the orchestration, as well as the semantics of their control constructs and data-flow. Ideally, the semantic descriptions of inputs and outputs should exist independently from the SWS specification. They should be chosen within existing ontologies describing a particular domain of knowledge and, at most, they might be refined. Hence, in order to define atomic SWSs just the semantic representation of conditions over inputs and outputs is needed, whereas to define composite SWSs the semantic representation of control constructs and data-flow is needed as well. Thus, Dynamic orchestration becomes mainly a representation problem. The SWS frameworks proposed in the literature provide different support to dynamic orchestration. In the next section we analyze the formal support offered by the two leading efforts for SWS representation (OWL-S and WSMO)[1].

2.2 WSMO and OWLS Formal Support

The Web Service Modeling Ontology (WSMO) provides a framework for semantic descriptions of Web services. It consists of four core elements that, properly linked, make up the semantic description of the service: **Ontologies**, **Goals**, **Web Services**, and **Mediators**. The semantics of these entities can be specified using one of the formal languages defined by the Web Service Modeling Language[2] (WSML). WSML includes several language variants, based on three different logical formalisms: Description Logics [1] (WSML-DL), First-Order Logic [4] (WSML-Full) and Logic Programming [9]

[1] Both OWL-S (`http://www.w3.org/Submission/OWL-S`) and WSMO (`http://www.w3.org/Submission/WSMO`) are currently submissions in the Semantic Web Services Standardization process.

[2] Web Service Modeling Language (WSML), W3C Member Submission, 3 June 2005. `http://www.w3.org/Submission/WSML/`



	Max usable expressive power combining the different types of SWS representation	
	WSMO	**OWL-S**
DL	WSML-DL (SHIQ(D))	OWL-DL (SHOIN(D))
LP	WSML-RULE (undecidable)	SWRL (undecidable)
DL & LP	WSML-Core (DL ∩ LP)	SWRL DL-Safe Rule (DL ∪ LP)

Fig. 2. OWL+SWRL vs. WSML formal support

(WSML-Rule and WSML-Flight). Furthermore, it defines the **WSML-Core** variant that can be identified as the intersection between a particular Description Logic (a subset of $\mathcal{SHIQ}$) and Horn Logic (without function symbols and equality) [6]. WSML-DL is, in general, incompatible with both WSML-Flight and WSML-Rule. Therefore, complete reasoning about service descriptions specified using WSML-DL, Rule and Flight is unfeasible because WSML-Full, the common super-language, is undecidable.

The Web Ontology Language for Services (OWL-S) is an OWL[3] ontology describing three essential aspects of a service (i.e., its advertisement, process model, and protocol details) using three different *modules*: **Service Profile**, **Service Model**, and **Service Grounding**. OWL-S itself is an OWL ontology. OWL, having formal foundations in Description Logics [1], provides three increasingly expressive sub-languages:

– **OWL-Lite**, that can be used to model classification hierarchies and simple constraints. OWL-Lite has the lowest computational complexity among OWL sub-languages.
– **OWL-DL**, used when the maximum decidable expressivity is required. It corresponds to $\mathcal{SHOIN}(\mathcal{D})$ DL [8].
– **OWL-Full**, although allowing for as much expressivity as RDF, is undecidable, and therefore it could be hardly be used for reasoning.

However, since (as will be pointed out in the following) the decidable fragment of SWRL [7] represents a reasonable advancement in reconciling Logic Programming and Description Logics, OWL-S is in a better position than WSMO as it allows to seamlessly integrate SWRL with OWL.

[3] OWL-S is based on OWL 1.0 recently extended with the new recommendation OWL 2, as reported in `http://www.w3.org/TR/owl-overview/`

2.3 OWL-S and WSMO in SWS Infrastructures

In this section we focus on the usage activities and service ontologies dimensions to compare and contrast the two proposed main framework: WSMO and OWL-S[4]. Indeed, the architecture dimension concerns the components needed to implement the use cases that do not affect the purposes of this comparison.

With respect to the *Usage activities* dimension, both WSMO and OWL-S take into account all the use cases that we have identified for dynamic orchestration. Their only difference is that WSMO requires to model the service requester features. This implies the need for tackling the interoperability problems between different WSMO elements within the framework itself. In order to overcome this issue, different types of mediators have been defined and considered in every usage activity, while OWL-S relies on ontology matching methods developed within the Semantic Web. This is reflected also in the *Service ontologies* dimension, where WSMO proposes a richer service description model than OWL-S. Consequently, some extra features related to Web service execution (like goals, mediator and communication protocols) need to be represented by WSMO model. The shortcoming of this approach is that any WSMO service usage is limited to its declared goals at design time.

The formal semantics offered by WSMO and OWL-S is fundamental to achieve *dynamic* orchestration. Figure 2 compares the formalisms on which the two frameworks are based. WSMO seems to be unrelated to the Semantic Web stack of languages and formalisms, except for its basic level (URI and XML), that guarantees syntactic interoperability on the Web. It offers mappings with the RDF syntax and the $\mathcal{SHIQ}$ Description Logic which, however, does not cover the whole OWL-DL. WSMO's global approach to service ontologies definition can be summarized as follows: it starts with the definition of a FOL-like language able to represent all the possible aspects of a Web service, then it defines several sub-languages restricting the original expressive power to well-known fragments (DL or LP), and (partially) maps such fragments onto current Semantic Web standard languages. Given the known incompatibility between DL and LP [2], currently the only implemented reasoning available for WSMO, when dealing with both rules and ontologies, is restricted to their common subfragment, i.e. DLP. Hence, as regards the implementation goal, to the best of our knowledge WSMO's inference capabilities are significantly reduced with respect to a pure Semantic Web based counterpart. OWL-S, instead, is based natively in OWL, whose natural extension towards rules is SWRL. SWRL's decidable fragment (DL-safe rules [11]) is the largest possible union of primitives from both DL and LP formalisms rather than their intersection. Therefore, from the perspective of two out of three main dimensions of SWS infrastructures, OWL-S turns out to be better equipped to encompass the requirements of *dynamic* orchestration.

[4] Semantic Annotations for WSDL and XML Schema (SAWDL) — W3C Recommendation, `http://www.w3.org/TR/sawsdl/` — has not been considered because it does not allows the process model representation.

3 OWL-S Support for Orchestration Dimensions

3.1 OWL-S Framework Details

OWL-S provides a Semantic Web Services framework to formalize an abstract description of a service. Since it is an upper ontology described with OWL, every described service maps onto an instance of this concept. The upper level **Service** class is associated with the following three other classes:

Service Profile. It specifies the functionality of a service. This concept is the top-level starting point for the customizations of the OWL-S model that supports the retrieval of suitable services based on their semantic description. It describes the service by providing several types of information: Human-readable information, Functionalities, Service parameters, Service categories.

Service Model. It exposes to clients how to use the service by detailing the semantic content of requests, the conditions under which particular outcomes will occur, and, where necessary, the step-by-step processes leading to those outcomes. It defines the concept *Process*, that describes the composition of one or more services in terms of their constituent processes. A *Process* can be *Atomic* (a description of a non-decomposable service that expects one message and returns one message in response), *Composite* (consisting of a set of processes within some control structure that defines a workflow) or *Simple* (used as an element of abstraction, i.e., may be used either to provide a view of a specialized way of using an atomic process, or a simplified representation of a composite process for reasoning purposes).

Service Grounding. A grounding is a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service. In general, a grounding indicates a communication protocol, a message format and other service-specific details.

Since this work aims at the automatic generation of services workflow and at its representation as an OWL-S composite process, more details about OWL-S process model are needed.

3.2 OWL-S Process Model

In this section we report the basic notions about the OWL-S process model with some considerations on the guidelines that should be followed in order to have useful metadata for the Web services to be described. Each OWL-S process is based on an IOPR (Inputs Outputs Preconditions Result) model. The *Inputs* represent the information required for executing the process. The *Outputs* represent the information the process returns to the requester. *Preconditions* are conditions that are imposed over the *Inputs* of the process and that must hold for the process to be successfully invoked. Since an OWL-S process may have several results with corresponding outputs, the *Result* entity of the IOPR model provides a means to specify this situation. Each result can be associated to a result condition, called *inCondition*, that specifies when that particular result can occur. Therefore, an *inCondition* binds inputs to the corresponding outputs. Such conditions are assumed to be mutually exclusive, so that only one result can be obtained for each possible situation. When an *inCondition* is satisfied, there are properties associated to this event that specify the corresponding output (*withOutput* property) and, possibly, the *Effects* (*hasEffect* properties) produced by the execution of the process. *Effects* are changes in the state of the world. The OWL-S conditions (*Preconditions*, *inConditions* and *Effects*) can be represented as SWRL logic formulas. Formally, *Input* and *Output* are subclasses of the more general class *Parameter*, declared in turn as a subclass of *Variable* in the SWRL ontology. Every parameter has a type, specified using a URI. Such a type is needed to refer it to an entity within the domain knowledge of the service. The type can be either a *Class* or a *Datatype* (i.e., a concrete domain object such as a string, a number, a date and so on) in the domain knowledge. Nevertheless, we argue that providing descriptions of Web service parameters using concrete datatypes adds very little semantics. For example, consider a service S whose input was declared as *Datatype* within a knowledge domain, e.g., a string. This means that the reference knowledge model of this input parameter is a concrete XML Schema datatype rather than an entity within a domain ontology. This mismatch becomes critical in automatic composition of services. Indeed, suppose that, during a hypothetical composition process, one needs to find another service whose output will be fed into S. Then, the composer must necessarily consider those services that have as output a resource of the same type as our input parameter, i.e. a string. Thus, any service that returns a string as an output can be composed with S, which would result in meaningless compositions of completely unrelated services due to the fact that the parameters have been poorly described form a semantic viewpoint. In the rest of this paper we will consider only those services having parameters declared as entities in a domain ontology (i.e., not as datatype). Furthermore, OWL-S Composite processes (decomposable into other Atomic or Composite processes) can be specified by means of the following *control constructs* offered by the language: **Sequence**, **Split**, **Split-Join**, **Any-Order**, **Choice**, **If-Then-Else**, **Iterate**, **Repeat-While** and **Repeat-Until**, and **AsProcess**. One crucial feature of a composite process is the specification of how its inputs are accepted by particular sub-processes, and how its various outputs are produced by particular sub-processes. Structures to specify the *Data Flow* and the *Variable Bindings* are needed. When defining processes using OWL-S, there are many places where the input to one process component is obtained as one of the outputs of a previous step, short-circuiting the normal transmission of data from service to client and back. For every different type of *Data Flow* a particular *Variable Binding* is given. Formally, two complementary conventions to specify *Data Flow* have been identified: **consumer-pull** (the source of a datum is specified at the point where it is used) and **producer-push** (the source of a datum is managed by a pseudo-step called *Produce*). Finally, we remark that a composite process can be considered as an atomic one using the OWL-S Simple process declaration. This allows to treat a Composite service as an Atomic one during the application of a Composer tool.

4 Automatic Composition for OWL-S

In this section we explain how to obtain OWL-S composite services using Semantic Web languages and tools. The proposed approach combines them with works presented in [14] and [13] endowing semantic interoperability. The procedure can be summarized as follows: 1) a SWRL representation is extracted from the available set of OWL-S atomic and simple services using a given encoding; 2) a SWRL composer generates a plan of rules that encodes the services; 3) the SWRL plan is interpreted to produce the OWL-S composite service.

4.1 Encoding OWL-S Atomic Processes with SWRL Rules

In this section we explain our approach for transforming process descriptions into sets of SWRL rules. SWRL [7] extends the set of OWL axioms to include Horn-like rules [9]. The proposed rules are in the form of an implication between an antecedent (*body*) and consequent (*head*), both consisting of a conjunction of zero or more atoms. The intended meaning can be read as: "whenever the conditions specified in the antecedent hold, the conditions specified in the consequent must also hold". For our purposes, it is important to highlight two SWRL characteristics: every rule must fulfil a *safety* condition (only variables that occur in the antecedent of a rule may occur in its consequent) and every rule with a conjunctive consequent can be transformed into multiple rules, each having an atomic consequent [9]. Furthermore, we work exclusively with SWRL DL-safe rules [11] fragment. Within OWL-S, conditions (logical formulas) can be declared using languages whose standard encoding is in XML, such as SWRL. Body and head are logical formulas, whereby OWL-S conditions can be identified with the body or with the head of a SWRL rule. Such conditions are expressed over *Input* and *Output*. Therefore, if the above requirement is met, conditions will be also expressed in terms of a domain ontology and thus will have the correct level of abstraction. After these considerations, we can describe the guidelines we follow for encoding an OWL-S process into SWRL.

– For every result of the process there exists an *inCondition* that expresses the binding between input variables and the particular result (output or effect) variables.
– Every *inCondition* related to a particular result will appear in the antecedent of each resulting rule, whilst the *Result* will appear in the consequent. An *inCondition* is valid if it contains all the variables appearing in the *Result*.
– If the *Result* contains an *Effect* made up of many atoms, the rule will be split into as many rules as the atoms. Each resulting rule will have the same inCondition as the antecedent and a single atom as the consequent.
– The *Preconditions* are conditions that must be true in order to execute the service. Since these conditions involve only the process *Inputs*, they will appear in the antecedent of each resulting rule together with *inConditions*. In this work we consider all the *Preconditions* as being always true.

The first guideline is needed because there may be processes in which the binding is implicit in their OWL-S descriptions. Let us consider, for example, an atomic process having a single output. In this case there might be no *inCondition* binding input and output variables since, being the output the only outcome, such a binding is obvious. This would prevent our encoding with SWRL rules because the second guideline would not be applicable. However, we can add a new *inCondition* that makes explicit such implicit binding. For example, suppose we have a service that returns book information, whose process is declared having one input (*?process:BookName*), one output (*?process:BookInfo*), and no condition. We should write the corresponding rule as "*kb:BookTitle(?process:BookName) $\rightarrow$ bibtex:Book(?process:BookInfo)*", but since variable *process:BookInfo* does not appear in the antecedent of the rule, this is not a valid SWRL rule. Since every service produces its output by manipulating the inputs, we may suppose that a *hasTransf* predicate exists, always true, that binds every input to the output. Adding this predicate to the rule antecedent we obtain the implicit *inCondition* and hence a valid rule. The entities that make up the SWRL rule (OWL Classes, properties and individuals) can be defined in different ontologies. For this reason we apply the matching of ontologies referred by the rules in order to enable semantic interoperability during composition. In particular, only the OWL classes need to be aligned, because properties in the rule are relations between considered classes and individuals are instances of considered classes. The matching procedure, which can be made by applying one of several learning techniques in the literature [5], produces equivalence assertions between classes. For example, consider the classes *books:Book* and *univ:Book*, where *books* and *univ* are the namespaces of two different ontologies *books.owl* and *univ-bench.owl*, respectively, describing the same domain. The result of the alignment will be an OWL axiom asserting that *books:Book* and *univ:Book* are equivalent classes. This axiom will be added to the knowledge base containing the SWRL rules.

4.2 The Composition Algorithm

Our SWRL composer prototype implements a backward search algorithm for the composition task and enhances the algorithm proposed in [14]. It works as follows: it takes as input a knowledge base containing SWRL rules (with the descriptions of the equivalent OWL classes) and a goal specified as a SWRL atom, and returns every possible path built by combining the available SWRL rules in order to achieve such a goal. These rules fulfil the SWRL safety condition. Specifically, the algorithm performs backward chaining starting from the goal in the same way as Prolog-like reasoners work for query answering. The difference is that this algorithm works on SWRL DL-safe rules instead of Horn clauses. This means that, besides the rule base, it takes into account also the Description Logic ontology the rules refer to. The SWRL rule path found, and consequently the resulting OWL-S service composition, will be valid (in the sense that it will produce results for the selected goal) only if the SWRL rules in the path are DL-safe. In other words, DL-safety means that rules are true for individuals that are *known*, i.e. that appear in the knowledge base[5]. The implemented prototype performs DL-safety check. This guarantees that the application of rules is grounded in the ABox and, consequently, that the services embodying those rules can be executed.

4.3 SWRL Plan Analysis

The set of paths obtained as a result of the composer can be considered as a SWRL rules plan (referred to as *plan* in the following) representing all possible combinable OWL-S

[5] It might not be the case in general, given the Open World Assumption holding in Description Logics, see [11] and Chapter 2 in [1].

Atomic processes that lead to the intended result (the goal). According to the OWL-S specification for a composed process and its syntax, the composition of atomic services obtained through the SWRL rule composer can be represented by means of an OWL-S composite service. In this section we will analyze a possible encoding. An OWL-S composed process can be considered as a tree whose nonterminal nodes are labeled with control constructs, each of which has children that are specified through the OWL property *components*. The leaves of the tree are invocations of the processes mentioned as instances of the OWL class Perform, a class that refers to the process to be performed. Bearing in mind the characteristics of the plan built by means of the method specified in Section 4.1, we identify the OWL-S control constructs to be used to implement the plan by applying the guidelines reported in Table 1. Currently, the OWL-S specification does not completely specify Iteration and its dependent constructs (Repeat-While and Repeat- Until), nor how the *asProcess* construct could be used. For this reason they are not discussed in this paper, but they will be considered in future work.

4.4 Encoding the SWRL Plan with OWL-S Constructs

For our purposes, each rule is represented in the composer as an object called *Rulebean*, that has various features and information that could be helpful to the composition, and specifically:

- The atoms in the declared precondition of the rule;
- The URI of the atomic process the rule refers to;
- The number of atoms in which the grafted rules have correspondence;
- A list containing pointers to the other Rulebeans with which it is linked.

The information about the atoms of the preconditions allows to check the presence of IF conditions that could lead to identify a situation that needs an If-Then-Else construct. The URI of the atomic process referred by the rule is needed because the leaves of the constructs tree must instantiate the processes to be performed. Finally, since each rule can be linked to other rules, it is necessary to store both their quantity and a pointer to the concatenated rules. In this way each Rulebean carries inside the entire underlying structure. This structure is implemented as a tree of lists, where each list contains the Rulebeans that are grafted on the same atom. Now let's show in detail the steps needed to encode a plan with the OWL-S control constructs. Referring to Figure 4, the procedure involves the application of three subsequent steps depending on the number n of grafted rules:

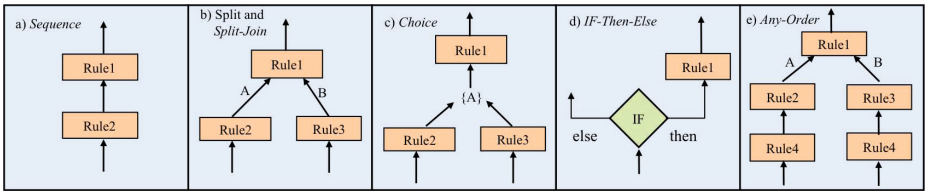


Fig. 3. The OWL-S control constructs in the plan

Table 1. OWL-S *Control Constructs* identified in the plan

Sequence. Represents the situation in which the rules are geared to each other sequentially, i.e. the head of a rule corresponds to an atom in the body of another one. Since this indicates a sequential execution, the Sequence construct will be used. According to the specification, *Sequence* is a construct whose components are executed in a given order and the result of the last element is used as the result of the whole sequence (Fig. 3 a)).
Split and Split-Join. Represents the situation where two or more rules with different head atoms are grafted directly into two or more atoms in the body of a particular rule. In this circumstance there is a branch that is evaluated and encoded with a Slit or Split-Join construct. According to the specifications, Split-Join is a construct whose components are executed simultaneously, i.e. they run concurrently and with a certain level of synchronization (Split is the particular case where synchronization is unnecessary). The condition for using this construct requires that its components can overlap in the execution, i.e. they are all different (Fig. 3 b)).
Choice. Represents the situation where two or more rules, with the same head atoms, are grafted directly into one of the atoms in the body of a particular rule. In this circumstance there is a branch that is evaluated and encoded with the Choice construct. According to the specifications, Choice is a construct whose components are part of a set from which any element can be called for execution. This construct is used because the results from the rules set can easily overlap, no matter which component is going to be run because the results are always of the same type (Fig. 3 c)).
If-Then-Else. It could represent the situation where the body of a rule are the atoms identifying a precondition. In this case, the service that identifies the rule to be properly executed needs that its precondition be true. In this circumstance, therefore, the precondition was extracted and used as a condition in the If-Then-Else construct. According to the specifications, If-Then-Else construct is divided into three parts: the 'then' part, the 'else' part and the 'condition' part. The semantics behind this construct is to be understood as: "if the 'condition' is satisfied, then run the 'then' part, otherwise run the 'else' part." (Fig. 3 d)).
Any-Order. Represents a situation similar to the Split-Join, but this particular case covers those circumstances where control constructs or processes are present multiple times in the structure of the plan, and it is important that their execution does not overlap in order to prevent a process break. This type of situation can be resolved through the use of the Any-Order construct because its components are all performed in a certain order but never concurrently (Fig. 3 e)).

1. Search all Rulebeans grafted with a number of rules equal to zero ($n = 0$) (Figure 4 a)).
   a. Store therein an object that represents the "leaf", i.e. an executable process.
2. Search all Rulebeans grafted with a number of rules equal to one ($n = 1$) (Figure 4 b).
   a. Check the engaged list:
      i. If there is only one Rulebean, the node will be of type "Sequence";
      ii. If there are multiple Rulebeans, the node will be of type "Choice";
   b. Store the object representing the created structure in the Rulebean.
3. Search all Rulebeans grafted with a number of rules greater than one ($n \geq 2$) (Figure 4 c).
   a. For each grafted list follow the steps in 2.a;
   b. Make the following checks on the structure:
      i. If there are repeated Rulebeans add a node of type "Any-Order";
      ii. If there are no repeated Rulebeans add a node of type "Split-Join";
   c. Store the object representing the created structure in the Rulebean.

Since the If-Then-Else construct overlaps with the constructs assigned during this procedure, it is identified in another way. During the creation of a Rulebean, a check is performed to verify if there are atoms in the body of the rule labeled as belonging to a precondition. In such a case, the Rulebean will be identified as the 'Then' part of the construct, and the atoms of the precondition will form the 'If' condition. The 'Else' part will be identified as the complementary path, if any (for the "Sequence" construct it does not exist, of course). Finally, the data flow is implemented in accordance with the consumer-pull method, i.e. the binding of variables is held exactly at the point in which it occurs.

Table 2. OWL-S Atomic services test set

Ws NAME	TEXTUAL DESCRIPTION	INPUTS	OUTPUTS
Service-10	This service returns the information of a book whose title best matches the given string	books:Title	books:Book
Service-12	A book search engine	books:Title	books:Book
Service-15	This service informs about a person who works as co-publisher of a certain book	books:Book	books:Person
Service-28	This service returns the author of the given novel	books:Novel	books:Person
Service-9	This service returns the price of a book; person is an optional input	books:Person; books:Book	concept:Price
Service-37	This service returns the name of books given the publication number	univ:Publication	univ:Book

5 Experimental Analysis

In this section we present an example that shows the applicability of our method. The considered services were chosen from the OWLS-TC dataset [6].

5.1 SWRL Plan Generation

Let us consider the subset of atomic services in Table 2. To obtain SWRL rules that satisfy the requirements described in Section 4.1, we have modified the atomic services as follows:

- For every parameter having a datatype as type, we create a class in the domain ontology having a datatype property with the corresponding datatype as range. The considered dataset did not require the application of this step.
- For each service, we create two logical formulas. The former consists of unary atoms having the *parameterType* URI as a predicate and the input as an argument, for each input. The latter consists of a unary atom having the *parameterType* URI as a predicate and the output as an argument. We set these two formulas as the antecedent and the consequent of a new SWRL rule, respectively.

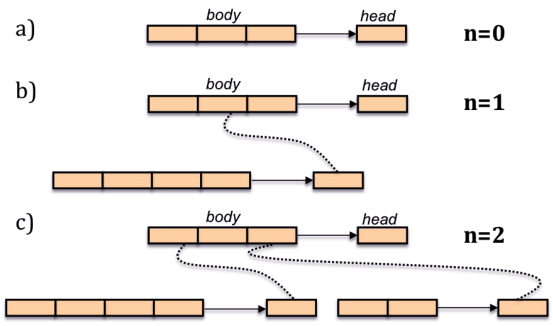


Fig. 4. The different types of grafted SWRL rules in the plan

[6] OWL-S service retrieval test collection,
`http://projects.semwebcentral.org/projects/owls-tc/`

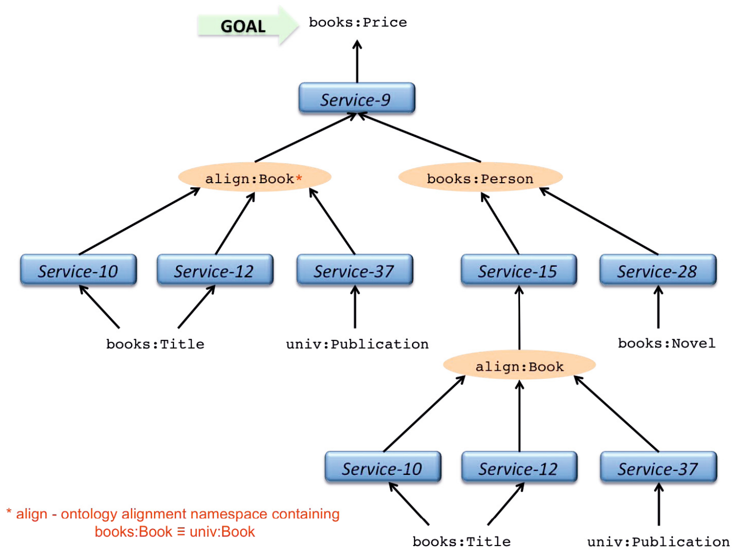


Fig. 5. The SWRL plan obtained by applying the composer to the services of Table 2

– Since every service produces its output by manipulating the inputs, we can suppose the existence of a *hasTransf* predicate, always true, that binds every input to the output, which guarantees the SWRL safety condition. Then we add *hasTransf* predicates to the antecedent of the rule built in the previous step. With this modification the antecedent can be identified with a new *inCondition*.

The SWRL rule set obtained in this way is given as input to the composer described in Section 4.2 and the resulting composition is shown in Figure 5.

5.2 OWL-S Composite Service Generation

After obtaining the composition plan, we apply the procedure described in Section 4.4. As a result we obtain the OWL-S constructs tree of Figure 6. In practice, we start searching Rulebeans grafted with a number of rules equal to zero, finding Service-10, Service-12, Service-37 (*books:Book* and *univ:Book* are equivalent classes) and Service-28. These will be tree leaves. We go on searching Rulebeans grafted with a number of rules equal to one, finding Service-15. It has two rulebeans grafted on the same Atom *books:Book*, thus we use a “Choice” construct. To link the obtained structure with Service-15 (another tree leaf) we use the “Sequence” construct, naming this structure $C1$. We continue serching Rulebeans grafted with a number of rules greater than one, finding Service-9. It has Service-10, Service-12 and Service-37 grafted on Atom *books:Book*, and Service-28 (another tree leaf) and $C1$ grafted on Atom *books:Person*. Both pairs are linked with a “Choice” construct, and we call them $C2$ and $C3$, respectively. Since $C2$ and $C3$ contain repeated Rulebeans (the “Choice” over Service-10,

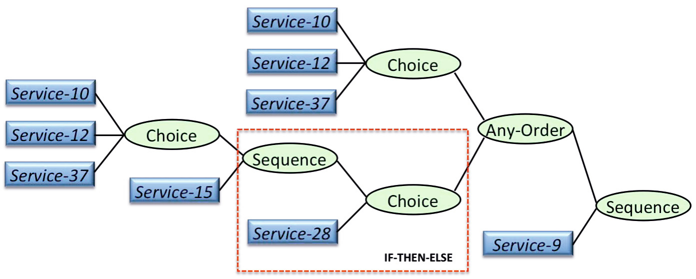


Fig. 6. The tree construct of the obtained composition

Service-12 and Service-37), we model this situation with the "Any-Order" construct rather than with the "Split-Join". The depicted "If-Then-Else" construct is obtained by applying the following consideration. Suppose that the precondition of Service-28 states that, in order to execute the service, the input must be necessarily a *books:Novel* (an OWL subclass of *books:Book*). Then, we can use this assertion as the 'If' condition, the execution of the service as the 'Then' part, and the non-execution of the service as the 'Else' part.

6 Conclusions and Future Work

The SWS frameworks proposed in the literature provide different support to dynamic orchestration. In the first part of this work we conducted a comparative study to elicit differences and analogies among them, and remarked the capabilities necessary to enable dynamic orchestration: the needed requirements and the suitability of OWL-S and WSMO to support such requirements. As shown in Section 2, the set of requirements is implied by means of the use cases, namely automatic discovery, selection, composition and invocation, required to make the SWS orchestration dynamic. The formal language underlying the SWS frameworks is the key for an effective realization of these use cases. For this reason, we described the formal support enabling reasoning on the semantic descriptions of the services offered by WSMO and OWL-S (based on OWL+SWRL and WSML, respectively). Then, we compared the formalisms underlying OWL+SWRL and WSML from the point of view of the expressive power actually exploitable for reasoning on service descriptions. As a result of this comparison, OWL-S turns out to be a more suitable candidate for *dynamic* orchestration. Automatic composition of SWSs is a more complex process to achieve using only tools built on Description Logics [1]. In Section 3 we presented a complete composition method for OWL-S services using Semantic Web languages and tools and working in semantically interoperable environments. The applied procedure can be summarized as follows: 1) The set of OWL-S atomic and simple (i.e., composite) services are represented by means of SWRL rules; 2) The SWRL composer is applied on them obtaining a plan of SWRL rules; 3) the SWRL plan is interpreted to produce the OWL-S composite service using OWL-S control constructs. The constructs identified in this way are used to build the OWL-S control construct tree that is directly serializable using the syntax of the language. As a future work, it is important to find ways to manage the remaining constructs (Iteration) and improve the composer in order to reuse internal parts of composite processes during composition. Another aspect that deserves attention concerns a Semantic Web intrinsic issue, i.e. the absence of primitives for retracting knowledge due to the monotonic nature of DL knowledge bases. Finally, the implementation of software agents able to manage dynamic SWS orchestration by considering low-level details as, for instance, Quality of Service, Service Level Agreement and the coordination for the concrete Web services Invocation would allows a large-scale use. As in previous work, the emphasis will be placed on the exclusive use of technologies developed for the Semantic Web.

Acknowledgments. Special thanks to Luigi Iannone, Terry R. Payne and Ignazio Palmisano for working with us in the definition and refinement of the paper themes.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge University Press (2003)
2. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82(1-2), 353–367 (1996)
3. Cabral, L., Domingue, J., Motta, E., Payne, T.R., Hakimpour, F.: Approaches to Semantic Web Services: an Overview and Comparisons. In: Bussler, C., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 225–239. Springer, Heidelberg (2004)
4. Enderton, H.B.: A Mathematical Introduction to Logic. Academic Press, New York (1972)
5. Euzenat, J., Shvaiko, P.: Ontology matching. Springer (2007)
6. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proceedings of the 12th International Conference on World Wide Web, pp. 48–57. ACM Press, New York (2003)
7. Horrocks, I., Patel-Schneider, P.F., Bechhofer, S., Tsarkov, D.: OWL rules: A proposal and prototype implementation. J. of Web Semantics 3(1), 23–40 (2005)
8. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: the making of a Web Ontology Language. J. Web Sem. 1(1), 7–26 (2003)
9. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer (1987)
10. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic Web Services. IEEE Intelligent Systems 16(2), 46–53 (2001)
11. Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with rules. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 3(1), 41–60 (2005)
12. Peltz, C.: Web Services Orchestration and Choreography. Computer 36(10), 46–52 (2003)
13. Redavid, D., Ferilli, S., Esposito, F.: SWRL Rules Plan Encoding with OWL-S Composite Services. In: Kryszkiewicz, M., Rybinski, H., Skowron, A., Raś, Z.W. (eds.) ISMIS 2011. LNCS, vol. 6804, pp. 476–482. Springer, Heidelberg (2011)
14. Redavid, D., Iannone, L., Payne, T.R., Semeraro, G.: OWL-S Atomic Services Composition with SWRL Rules. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) ISMIS 2008. LNCS (LNAI), vol. 4994, pp. 605–611. Springer, Heidelberg (2008)

Agent-Based Framework Facilitating Component-Based Implementation of Distributed Computational Intelligence Systems

Kamil Piętak and Marek Kisiel-Dorohinicki

AGH University of Science and Technology, Kraków, Poland
{kpietak,doroh}@agh.edu.pl

Abstract. The paper presents a framework particularly suitable for the design of a certain class of distributed computational intelligence systems based on the agent paradigm. A starting point constitutes a formalism utilizing the notions of algorithms and dependencies, which allows for the formulation of the system functional integrity conditions. Next, technological assumptions of *AgE* framework are presented and a direct mapping between the formalism and the implementation structure of the framework is discussed. The approach assumes that component techniques facilitate the realization of the particular system in such a way that algorithm dependencies are represented as contracts. These allow to support the verification of the system's functional integrity. Selected technical aspects of the framework design illustrate the considerations of the paper.

1 Introduction

Agent technology aims to provide concepts and tools for the development of complex, decentralized systems [14]. Of course a multi-agent system may be implemented without any structures or services specific to agents. This often happens for simulation systems, where the introduction of agents facilitates modeling of complex phenomena – natural, social, etc. In such cases agents constitute building blocks of the model, which may or may not be implemented with the use of agent technology. But since the key concept in multi-agent systems constitute intelligent interactions, agents must be able to cooperate with one another, which requires adequate infrastructure addressing interoperability issues [2].

The paper focuses on a specific class of multi-agent systems, which use computational intelligence paradigms – particularly hybrid techniques based on the concept of decentralized evolutionary computation [9]. These systems consist of a large number of agents performing similar tasks, but working with different structures and mechanisms. Thus from the software engineering perspective it may be said that the system is decomposed into particular agents, but a single agent implementation is too complex to serve as an assembly unit. In fact agents may be further decomposed into functional parts (components), which are replaceable, as long as they are compatible to one another, even when used by different agents (this ensures agents interoperability at implementation level).

As it was discussed in [2], both agents and components can be considered in software development as assembly units, which implement in various ways the concept of responsibility delegation. However, agent-based technology focuses more on executing complex tasks in a community to achieve defined goals and, on the other side, component-based technology is rather aimed at reusability and integrity aspects of software development [10]. Indeed, it seems that component-oriented approach can be also successfully exploited in agent-based systems for assembling parts of agents implementation and checking the integrity of the system.

To facilitate the design of the systems under consideration a dedicated formalism was proposed in [4], and further developed in [11], demonstrating how the system *functional integrity* conditions may be supported by component technology. The approach is based on the notion of algorithms used by agents for performing actions, and dependency relation imposing the existence of algorithms required by actions, as well as other algorithms. Agents and algorithms are implemented and provided to *AgE* framework as components, which supports the assembly of particular systems at run time based on the provided configuration. The goal of this paper is to show how the relations between agents and algorithms are implemented as component contracts handled by the configuration engine of *AgE* framework, which is also responsible for the verification of the system functional integrity.

The paper begins with the review of a formalism, which allows for the formulation of functional integrity conditions of the system. The design issues of *AgE* framework, as well as the mapping between the formalism and the implementation structure are described in the next sections. Finally, the assumptions of *AgE* framework component model are introduced, followed by the presentation of selected realization issues using component techniques.

2 The Model of a Computing MAS

The model proposed in [4] defines an agent as a tuple:

$$AG \ni ag = \langle id, tp, dat_1, \ldots, dat_n \rangle \tag{1}$$

where $id \in ID$ is a unique identifier of an agent[1], $tp \in TP$ denotes the type of an agent (depending on its type, an agent is equipped with specific data and may perform specific actions), and $dat_i \in DAT_i, i = 1, \ldots, n$ represents problem-dependent data (knowledge) gathered by an agent.

According to [4] a multi-agent system includes agents, actions to be executed by the agents, and the environment represented by some common data, which may acquired by the agents. This definition must be extended here with a set available algorithms:

$$AS \ni as = \langle Ag, Act, Alg, qr_1, \ldots, qr_m \rangle \tag{2}$$

[1] For each element of the model its domain, which is a finite set of possible values, is denoted by the same symbolic name in upper case, e.g. *ID* is the set of all possible agent identifiers.

where $Ag \subset AG$ is the set of agents of as, $Act \subset ACT$ describes actions that may be performed by the agents of as, $Alg \subset ALG$ is the set of algorithms available in as, and $qr_i \in QR_i, i = 1, \ldots, m$ denote queries providing data (knowledge) available for all agents in as.

In the space of types, a subsumption relation "$\preceq$" $\subset$ $TP \times TP$ is defined, introducing a partial order in TP. In terms of this relation, $A \preceq B : A, B \in TP$ means that A is a subtype of B.

Agents may perform actions in order to change the state of the system. An action is defined as the following tuple (Hoare's triple equivalent [13]):

$$ACT \ni act = \langle tp, pre, post \rangle \tag{3}$$

where $tp \in TP$ denotes the type of agents allowed to execute the action (only agents of the type tp and descendant types – according to the "$\preceq$" relation – may perform the action); $pre \in X$ is the state of the system which allows for performing action act; $post \in X \times X$ is the relation between the state of the system before and after performing action act.

Actions may depend on algorithms, i.e. in order to perform an action, one or more algorithms may be needed. This dependency is described by the following relation:

$$\text{“}\rightsquigarrow\text{”} \subset ACT \times ALG \tag{4}$$

For algorithms there is also a subsumption relation "$\preceq$" defined, which states whether one algorithm is a specialization of another:

$$\text{“}\preceq\text{”} \subset ALG \times ALG \tag{5}$$

Relation "$\preceq$" introduces a partial order in ALG (it is reflexive, transitive and antisymmetric), i.e. if $A1 \preceq A$ then $A1$ can be used in place of A when needed.

When an action is about to be performed, a subset of algorithms is selected from Alg according to the $\preceq$ relation and any further restrictions described below. These algorithms are said to be "available" in the environment for the execution of a particular action. For example, if action act depends on algorithm A, and in a particular system $\exists!\, subA \in Alg : subA \preceq A$, then when the action is executed, it uses algorithm $subA$.

Further dependencies between algorithms used in the system may be described using the following relation:

$$\text{“}\rightsquigarrow\text{”} \subset ALG \times ALG \tag{6}$$

If algorithm A depends on algorithm B ($A \rightsquigarrow B$) it means that B or its subtype is needed for A to function properly, and must be available in the system. Relation "$\rightsquigarrow$" allows for defining families of algorithms that are designed to be used together (i.e. if one of them is selected by the environment for the execution of a particular action, then other algorithms from the same family are also selected).

The proposed formalism allows to formulate the conditions of *functional integrity* of the whole system [11]. The notion of *functional integrity* may be understood as the ability to fulfill functional requirements in a complex system.

According to assumptions of the discussed model the system is functionally integral when the following coherency conditions are true for all agent subsystems $AS \ni as = \langle Ag, Act, Alg, qr_1, \ldots, qr_m \rangle$:

$$\forall\, act \in Act\, [\, (\exists\, alg \in ALG : act \rightsquigarrow alg) \Rightarrow (\exists\, alg_c \in Alg : alg_c \preceq alg)\,] \qquad (7)$$

$$\forall\, alg_1 \in Alg\, [\, (\exists\, alg_2 \in ALG : alg_1 \rightsquigarrow alg_2) \Rightarrow (\exists\, alg_g \in Alg : alg_g \preceq alg_2)\,] \qquad (8)$$

i.e. for each action that may be performed in the agent system ($act \in Act$), if this action depends on algorithm alg, then an algorithm alg_c subsuming alg must be available in the system (i.e. must be present in the Alg set of the system). As was mentioned before, the subsumption relation "$\preceq$" is a partial order, and therefore alg_c can equal alg because $alg \preceq alg$. Similarly, for each algorithm alg_1 that is available in the AS, if alg_1 depends on alg_2, then an algorithm subsuming alg_2 (in particular, alg_2 itself) must be also available in the agent system ($\exists\, alg_g \in Alg$).

The case of modeling an evolutionary multi-agent system [8] concerning its functional integrity conditions is discussed as an illustration in [11].

3 AgE Computing Framework

The model presented is the base for the design of the core of the computing framework *AgE*[2], which is developed as an open-source project at the Intelligent Information Systems Group of AGH-UST. The framework is designed to support the construction of a wide range of computational systems build according to agent paradigm, with special attention paid to collective computational systems [4]. The goal of the project is to provide a platform allowing for flexible (re)configuration of different system variants to meet the requirements of particular problems and solving techniques.

The framework assumes that a particular system is constructed by providing its configuration (e.g. in XML format or generated by some designer tool—see figure 1). The configuration specifies the system structure and required parameters. After start-up, the configuration is distributed amongst available nodes, where the required components are instantiated and initialized. If needed, additional services such as name, monitoring, communication and topology services are also run on selected nodes to manage and monitor the computation. A computation is logically decomposed into agents which are responsible for performing parts of or the whole algorithm. Agents co-operate in the local environment provided by so-called *workplaces*, which are distributed amongst many nodes (figure 2) according to the algorithm decomposition.

Each agent has an unique address which identifies it across the whole system. The address is registered in a name service which tracks information about current agent locations. With the use of the environment, agents can communicate with their neighbor agents via messages or queries or request them to perform specific actions.

[2] `http://age.iisg.agh.edu.pl/`

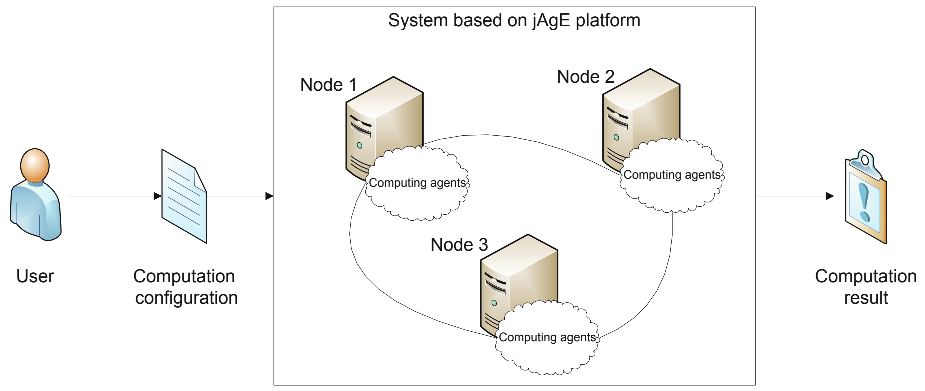


Fig. 1. AgE functional schema

The platform introduces two types of agents: thread-based and simple. The former are realized as separate threads so that the parallel processing is managed by Java Virtual Machine (similarly to eg. *Jade* platform[3]). Such agents can communicate and interact with neighbors via asynchronous messages.

However, a large number of such agents would significantly decrease the performance and efficiency of a computation because of frequent context switching, therefore the notion of simple agent was introduced. The concept of simple agent is realized as event-driven simulation, which allows pseudo-parallel execution of agents' tasks. Two phases are distinguished:

– Execution of tasks related to computation semantics in the `step()` method. While doing so, they can register in the workplace various events, which may indicate actions to be performed or messages to be sent.
– Processing of events registered in an event queue performed by the workplace.

The described idea of agents processing ensures that during the execution of computational tasks of agents co-existing in the same workplace, the environment remains unmodified, thus the tasks may be carried out in any order. From these agents perspective, they are processed in parallel. All changes to the agent structure are made by workplaces during processing of the events indicating actions such as addition of new agent, migration of an agent, killing an already existing agent, etc. Their results are observable for agents while performing the next step.

Each agent exists in an environment, defined by the workplace, which provides a context of agent's processing. This means that the environment is responsible for communication between agents and for responding to queries from

[3] `http://jade.tilab.com/`

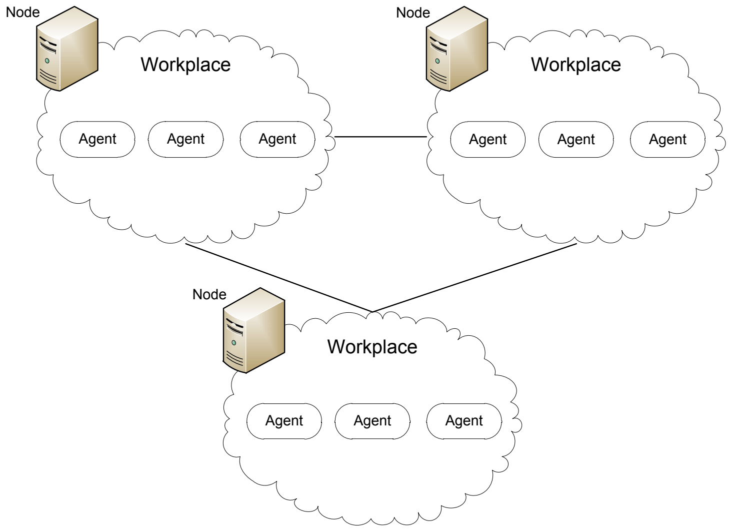


Fig. 2. Workplaces and agents

agents. What is more, according to the presented concept, the environment determines the types of actions which are available for simple agents. It also provides concrete implementations of these actions and thereby supplies and influences agents' execution. The actions can be executed via workplaces methods or external strategies specified in the input configuration. Actions realize the principle of goal-level communication [2], because agents only let the environment know what they expect to be done but they do not know how it is done. The decision of how to execute the action is made by the workplace which resolves proper action implementation according to *service locator* design pattern [1].

4 AgE Agents Implementation Structure

Following the assumptions of the model, *AgE* agents are not atomic assembly units, but they are further decomposed into functional units (algorithms and actions) according to *strategy* design pattern [5]. Strategies represent specific operators, which may be exchanged without intruding agents' core implementation. Their instances may be shared between agents as they provide various services to agents or other strategies.

Both strategies and agents can have named properties, which represent their features. These can be referenced during run-time by their names in order to access, modify or even monitor their values. Properties may be accessed via methods or fields. In the former case the accessor method has to be annotated with the use of `@PropertyGetter` tag and, if the property may be modified, the mutator method has to be annotated with corresponding `@PropertySetter` tag. In case of field-based properties, only the field has to be annotated with `@PropertyField` tag—this type of properties is always modifiable. For each class a list of its annotated properties may be retrieved and each named property of its instance may be accessed in a uniform way. The properties may be of simple as well as reference types. The mechanism of properties was described in more detail in [4].

Agents and strategies should be loosely coupled. First of all it means that their dependencies should be realized through well-defined interfaces and not by concrete implementations. This not only hides implementation details but also allows for changing dependencies without recompilation of involved classes. This is essential in terms of flexible component assembly described in the next sections. Moreover, agents and strategies can have simple parameters, which may be used to tune the algorithms for specific problems. Both dependencies and parameter values may be set by the framework based on the supplied configuration with the use of properties.

Actions are implemented and executed as methods of external strategies classes or the workplace class, which represents the agent's environment. During the execution of these methods other strategies can be used to perform different activities within the action. Therefore, the dependency (described by the relation "$\rightsquigarrow$") between actions and algorithms is represented in *AgE* as a reference property to the required algorithm, preceded by @Inject annotation.

Dependency between algorithms introduced in the model maps to dependency between strategies. In different cases this relation can be represented in two ways in *AgE*. Let us consider dependent algorithms A and B ($A \rightsquigarrow B$) represented in *AgE* by Java classes `A` and `B`. In the first case strategy `A` directly uses strategy `B` and the dependency relation is realized in the same way that dependency between actions and algorithms, i.e. by using a reference property marked with @Inject annotation. In the next case, strategy `A` does not directly use `B`, but `B` is required for proper processing of `A`. For example, a binary mutation strategy needs a binary initialization strategy which generates the proper type of solutions, although the initialization is not directly used by mutation. To define this kind of dependency, class `A` must be preceded by annotation @Require(B.**class**). An example code of a strategy with dependencies is shown below:

```
@Require(C.class)
public class A {
    @Inject
    private B b;
}
```

Strategy `A` directly uses strategy `B` and requires strategy `C` for proper processing. Therefore it defines two dependencies, the first by an annotated reference property (for strategy `B`) and the second by adding @Require annotation to class definition (for strategy `C`).

5 Component Techniques for AgE Framework

AgE was designed with the emphasis on achieving the main advantages of component-oriented techniques, i.e. independent development, reusability and elementary contracts [13]. Its realization is vastly supported by *dependency injection* pattern [12], an implementation of the inversion of control paradigm popularized by Martin Fowler[4], and by utilizing the freely available PicoContainer framework[5].

The main composable part managed by the framework is *a component* represented by a single class with additional information about its capabilities and requirements. A system or a part of it is assembled using IoC container[6] based on the run-time configuration, which describes inter alia bindings between components. In run-time environment components are instantiated as objects and in this shape they realize their responsibilities.

Component Descriptor

A component is described by *a component descriptor*, which contains the component identifier (i.e. a component class name) and details about the provided interfaces, dependencies and properties. It is read from the declarative description and allows for further component processing in the run-time environment. Specifying the descriptor as an abstract notion, the component model is not bound to a specific implementation of the component description. One can specify component using Java annotations (for example annotations proposed in JSR-330 [7]), applying conventions (such as Java Beans [6]) or even by providing external file in a textual format such as XML.

Provided interfaces are defined as interfaces realized by a component class. It is assumed that each interface defines a set of public, well-documented methods, which constitute the functionality offered by the component. *Component dependencies* are expressed as a set of named dependencies to required interfaces. In the run-time environment a dependency is realized as a reference to the component instance which provides a required interface. Also, one-to-many dependencies are supported—they are implemented as a reference to a collection of dependant instances. Declared dependencies are injected while creating a new component instance by an IoC container.

The last part of the component description is a declaration of *component properties*. The properties give an opportunity to provide context information to the component instance. They are realized as object attributes of simple types such as *int*, *double*, or *String*. The list of the available property types is not fixed and can be extended by providing a simple type interpreter to the framework.

[4] `http://martinfowler.com/articles/injection.html`

[5] `http://www.picocontainer.org`

[6] Inversion-of-Control container that realizes dependency injection pattern.

Runtime Configuration

A *run-time configuration* is represented as a set of *component definitions*. Each definition can be perceived as a recipe for a component instantiation. The configuration contains definitions of components used in a particular system and specifies *bindings* between components. To gain more control over the process of the system composition, the context introduces a new name space for components, based on textual identifiers (called *component names*). This gives a possibility to define many different configurations for one component, simply by specifying component definitions with different names, but attached to one component class.

The bindings defined in the component definition tell that a given dependency declaration is bound to a concrete component, which offers the required interface. The bindings may be specified explicitly and implicitly. In the first option, a component dependency is assigned to a concrete component name, defined within the same context. On the other side, implicit bindings are not expressed in the configuration. While instantiating component the framework tries to resolve the dependency by searching a component instance, which offers an interface defined in the dependency declaration.

Finally the component definition may also specify initial values for component properties.

The configuration can be created based on external sources such as XML files, container API, or dedicated domain specific languages.

Verification Mechanism

The configuration expressed as a set of component definitions gives the container a full knowledge of all components that can be instantiated. This knowledge can be further used to perform a priori verification before any object is produced be the container. Otherwise, a system may run normally until the moment of using an incorrectly configured component. Such late error detection may have serious consequences for the system.

Concerning component dependencies, there are different types of problems that should be detected:

– *lack of component dependency* occurs when the component dependency is required but it is not available for the container,
– *inconsistent type of component dependency* occurs when the required dependency type is different in component descriptor and in the binding; this may occur when the container configuration is provided in external files where it is impossible to check this in compile-time,
– *cyclic dependency* occurs when a component is dependent on another component, which is dependent on the first one and both of them cannot be created without each other.

All these cases can be checked by the analysis of component definitions and relevant component descriptors in the context of the current environment (i.e.

IoC container). For example, the verification mechanism may check if a binding defined in a component definition, points to a component with type compliant with the type defined in the component descriptor.

The verification mechanism ensures the functional integrity of the system as defined by (7) and (8)—all bindings are processed in order to check if all required components are available in the class-path, no conflicts occur between components, and all requirements are fulfilled.

6 Selected Realization Aspects of Component Dependencies and Their Verification

As it was mentioned before, the proposed solution is based on PicoContainer framework as it may be easily customized and extended. It provides an implementation of IoC container (`DefaultPicoContainer`), which is a registry of components and is responsible for providing component instances. The internal structure of PicoContainer is modular and assumes that a request for retrieving a component instance is delegated to the appropriate `ComponentAdapter` object, which is associated with each single type of component.

Since *AgE* operates on two types of logical units: stateful (agents) and stateless (algorithms), both may be provided to the framework as components. Adapters can be decorated in order to add new features, such as instance caching, which is required for providing shared instances of stateless components. By implementing a custom `ComponentAdapter` one can modify and adjust the process of building component instances. This allows *AgE* components to define named properties (as described in the previous section), which serve as dependencies specification.

Initialization of the Component Framework

The process of initialization of the component framework is divided into three main phases.

1. An input configuration is read from XML file with well-defined structure[7] and further transformed into the object model of runtime configuration comprised of `ComponentDefinition` instances. Each definition describes a configuration of a single component and contains data such as component name, type (which is the name of a class) and scope, which is to determine if a new component is to be created for each request (*prototype* scope) or only once during the first request (*singleton* scope). The definition also specifies constructor arguments, as well as property initializers, responsible for providing information on how to set reference or simple values for particular component properties. Moreover, the definition object contains `createInstance` method, which is used to instantiate a described component with initialized dependencies (this process is described in more detail in the next subsection).

[7] `http://agh.iisg.agh.edu.pl/age-2.3.xsd`

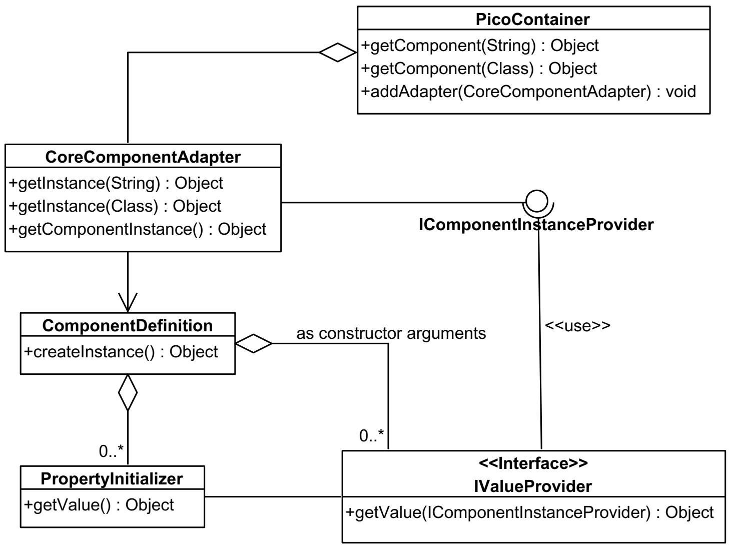

PicoContainer
+getComponent(String) : Object
+getComponent(Class) : Object
+addAdapter(CoreComponentAdapter) : void
CoreComponentAdapter
+getInstance(String) : Object
+getInstance(Class) : Object
+getComponentInstance() : Object
IComponentInstanceProvider
<<use>>
ComponentDefinition
+createInstance() : Object
as constructor arguments
0..*
0..*
PropertyInitializer
+getValue() : Object
<<Interface>>
IValueProvider
+getValue(IComponentInstanceProvider) : Object

Fig. 3. Dependency Injection in *AgE* framework

2. An instance of IoC container is created and initialized with component definitions. For each definition a dedicated adapter (`CoreComponentAdapter`) is created and registered in a container as shown in figure 3. Also, the adapter implements `IComponentInstanceProvider` interface, which defines methods for explicit retrieving instances of components by name or type.
3. The validation of the given configuration is performed. It allows for detecting errors such as unresolved dependencies, non-existent components or incorrect property definitions before any component instance is created. The mechanism extends PicoContainer concept of validation based on *visitor* design pattern [5]. As shown in figure 4 the proposed solution provides `Verifier` class that extends `VerifingVisitor` defined by PicoContainer framework. An instance of `Verifier` traverses every component definition registered in the IoC Container via `CoreComponentAdapter`. While visiting the definition, it verifies the dependencies according to the `ComponentDescriptor` instance relevant to the component class. Any errors are collected and returned after finishing the process.

Creating Component Instances

When the container receives a request for a component instance, it locates the appropriate component adapter (using given name or type) and delegates this request to the adapter. The adapter calls the associated component definition's `createInstance` method, which is responsible for creating a component instance. The method creates a new object using the proper constructor (based on

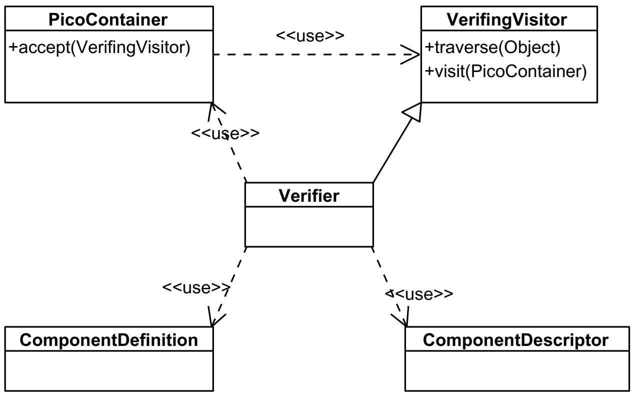


Fig. 4. Verification mechanism utilizing *visitor* design pattern

the specified arguments) and than iterates through all properties and initializes them according to the definition. During the initialization of both, constructor arguments and properties, an `IComponentInstanceProvider` instance is used to fetch references to dependent components instances. The provider, which is in fact the component adapter itself, retrieves component instances from the IoC container, and the process starts again. In the case of simple types, a value is kept directly in a value provider object and is returned on a request. The whole process is repeated until all dependencies are resolved and then the fully-initialized component instance is returned to the client.

A component class which implements `IComponentProviderAware` interface is provided with `IComponentInstanceProvider` that created it. This gives such a component the ability to retrieve other component instances at run-time via the same provider according to *service locator* design pattern [1].

7 Practical Evaluation

The framework proved to be a convenient and flexible tool supporting the assembly of a wide range of simulation and computing systems. As an example, two different evolutionary optimization approaches may be considered: classical distributed genetic algorithms and evolutionary multi-agent systems [8]. The idea of EMAS was proposed as a particular technique of decentralized evolutionary computation. The system consists of individual agents, which possess solutions of the given optimization problem. They also possess a non-renewable resource called *life energy*, which is the base of a distributed selection process. The application of both classical GA and EMAS to solving concrete optimization problems requires among other things the choice of solution representation and adequate variation operators. Obviously these components may be easily exchanged for different application areas, only if they are represented by common interfaces [4]. At the same time both classes of systems may be built from the same problem-oriented components, which may be used by different agents implementations (populations in classical GA or individuals in EMAS).

Yet the approach seems to be especially suitable for far more complex systems. This is especially important since today many results in the field of *computational intelligence* are based on combining different ideas and methods, which by the effect of synergy exhibit some kind of intelligent behaviour. Such approach represents an agent-based system for evolutionary optimization of the architecture of a predicting neural network with an immune-inspired selection mechanism, which was also successfully realised with the use of *AgE* [3].

8 Conclusions

AgE framework discussed in the paper was designed to support the assembly of agent-based systems from components representing not only agents but also their functional parts (so-called algorithms) according to the provided configuration. The framework provides validation mechanism with respect to the proposed functional integrity conditions, which impose the existence of algorithms required by agents and other algorithms. This is especially important in distributed environments, since in such cases the validation is performed independently on each node based on local resources.

It was shown how component-oriented techniques may be exploited to realize the process of automatic assembly of different agents structures with dependent strategies. The configuration together with appropriately annotated agents and strategies classes allow for creation of fully-initialized components. Moreover, late binding by a container allows for run-time injection, which facilitates third-party development of the components. The annotations describing component dependencies, together with public methods of the class treated as component's operations, may be perceived as a requirement closely related to component contracts as proposed by Szyperski [13]. The decision, which component should be provided to the other one, is made during instantiating particular components based on the run-time configuration. The functional integrity of the system is ensured by verifying the configuration and components descriptors—dependencies are processed in order to check if all required components are available in the classpath, no conflicts occur between components, and all requirements are fulfilled. Verification is also performed during each request for instantiating a component, which ensures that no inconsistent unit will be created.

Particular systems are built by providing appropriate configurations that instruct the framework how to assembly a complete solution from available components, as it was illustrated for the considered evolutionary optimization systems. In this case, independent parts of the system (such as solution representation, operators, selection mechanism, etc.) can be easily modified by updating existing configurations. This can be very helpful for example during process of adjusting algorithms configurations for new problems. The component-based approach gives also an opportunity to create new systems without awareness of implementation details—one needs only to know available interfaces provided by available components and understand dependencies between them.

The main thread of further research in the considered area concerns the support for components migration, which might be used to dynamically configure the distributed nodes' run-time environments, as well as an extension to agent migration and inter-agent communication infrastructure.

Acknowledgments. The work presented in this paper was partially supported by the Polish Ministry of Science and Higher Education grant No. N516 500039.

References

1. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall (2003)
2. Bergenti, F., Gleizes, M., Zambonelli, F.: Methodologies and Software Engineering for Agent Systems. Kluwer Academic Publisher (2004)
3. Byrski, A., Kisiel-Dorohinicki, M.: Immune-based optimization of predicting neural networks. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005, Part III. LNCS, vol. 3516, pp. 703–710. Springer, Heidelberg (2005)
4. Byrski, A., Kisiel-Dorohinicki, M.: Agent-based model and computing environment facilitating the development of distributed computational intelligence systems. In: Allen, G., Nabrzyski, J., Seidel, E., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2009, Part II. LNCS, vol. 5545, pp. 865–874. Springer, Heidelberg (2009)
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
6. Java Specification Request 220: Enterprise JavaBeans 3.0 (2006), `http://jcp.org/en/jsr/detail?id=220`
7. Java Specification Request 330: Dependency Injection for Java (2009), `http://jcp.org/en/jsr/detail?id=330`
8. Kisiel-Dorohinicki, M.: Agent-based models and platforms for parallel evolutionary algorithms. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004, Part III. LNCS, vol. 3038, pp. 646–653. Springer, Heidelberg (2004)
9. Kisiel-Dorohinicki, M., Dobrowolski, G., Nawarecki, E.: Agent populations as computational intelligence. In: Rutkowski, L., Kacprzyk, J. (eds.) Neural Networks and Soft Computing, Physica-Verlag (2003)
10. Krutisch, R., Meier, P., Wirsing, M.: The agentComponent approach, combining agents, and components. In: Schillo, M., Klusch, M., Müller, J., Tianfield, H. (eds.) MATES 2003. LNCS (LNAI), vol. 2831, pp. 1–12. Springer, Heidelberg (2003)
11. Piętak, K., Woś, A., Byrski, A., Kisiel-Dorohinicki, M.: Functional integrity of multi-agent computational system supported by component-based implementation. In: Mařík, V., Strasser, T., Zoitl, A. (eds.) HoloMAS 2009. LNCS (LNAI), vol. 5696, pp. 82–91. Springer, Heidelberg (2009)
12. Prasanna, D.R.: Dependency Injection, Design patterns using Spring and Guice. Manning Publications Co. (2009)
13. Szyperski, C.: Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
14. Wooldridge, M.: An Introduction to Multiagent Systems. John Wiley & Sons (2009)

A Hardware Collective Intelligence Agent

Kin Fun Li and Darshika G. Perera

University of Victoria, Victoria BC V8W 3P6, Canada
{kinli,darshika}@uvic.ca,
http://www.ece.uvic.ca/~kinli

Abstract. In recent years, several powerful computing models including grid, cloud, and Internet have emerged. These state-of-the-art paradigms offer numerous benefits to large-scale and compute-intensive applications such as data analysis and decision modelling in enterprise systems. Many of these applications make use of the collective intelligence technique due to the necessity in a widely dispersed environment, or the desirability to harness the processing power and aggregated knowledge in a distributed system. Most current implementations of the intelligent agent model are software based. This work proposes the use of hardware collective intelligence agent in lieu of the software version, in order to achieve flexibility, versatility, and scalability. Housing on a single chip, the hardware agent is useful in the emulation of collective intelligence models, and deployment in realistic collaborative settings. The rationales of using hardware agent, its advantages, and performance are presented, discussed, and analysed.

Keywords: collective intelligence agent, hardware computation models, emulation, FPGAs, reconfigurable computing.

1 Background and Introduction

Collective Intelligence (CI) is a common theme in many prominent research areas such as social network, artificial intelligence, and Web mining. Multiple intelligent agents interact with each other contributing to a common goal. These autonomous agents, often referred to as situation agents or adaptive agents, have the capability to learn from each other as well as by themselves. The CI paradigm enables decision-making collectively using global knowledge rather than based on local information. Flexibility, extensibility, and scalability of each agent are the major factors leading to successful implementation of a CI system. A prime example of success is Wikipedia for its great achievement in building knowledge collectively.

Pentland carried out an extensive empirical study in [15] and concluded that "human intelligence is substantially, and perhaps even largely a collective, network phenomenon." According to Gregg [7], "a collective intelligence application is one that harnesses the knowledge and work of its users to provide the data for the application and to improve its usefulness." CI can be found in numerous application domains including recommender systems [20], Web business intelligence [9], business processes and models [3], and collaborative learning [11].

An emergency response system designed by Vivacqua and Borges [23] focuses on harnessing CI from ‘crowdsourcing’ (i.e., outsourcing to a crowd instead of doing it in-house by a few individuals). Rasmussen et al. proposed Internet based CI methods to support future energy system decision making by large stakeholder groups in [19]. They emphasized the need for CI to accommodate large-scale systems. This is a necessary requirement for any CI system to function as it often utilizes large-scale state-of-art computing platforms including Internet, grid, and cloud.

1.1 CI Agent Modelling, Development and Implementation

Agent-based models and simulation techniques are suitable vehicles to study social systems and CI, as advocated by Singh and Gupta in [21]. Vergados et al. presented a framework for engineering CI systems to be used by Web communities in [22]; however, they relied on a simulation model to evaluate their proposal, citing the impractical nature of post-deployment evaluation.

Almost all of the CI simulation studies and applications are implemented in software, for various collaborative network and agent-based distributed environments. An exception is the on-chip multiprocessor design based on swarm intelligence proposed by Narasimhan et al. in [13]. This shared-memory, tightly-coupled multiprocessor has its processing elements organized in a 2D array. This design was evaluated in a simulator with signal processing applications.

1.2 Motivation of a Hardware Approach

A couple of conclusions can be drawn from this brief survey and discussion, that are important for the advancement of CI technologies. The first is speed performance, for which one can utilize the readily available parallel processing techniques such as cluster and grid. Efficient hardware in any form, therefore, plays a key role in this essential requirement. The second is the proper and accurate simulation of a proposed model before actual deployment. However, recent trend has shifted to large-scale emulation using thousand-plus machines to deal with the complexity of real-world system behaviours, that simulation may fail due to the many assumptions made [14]. This need of at-scale emulation is exemplified by the promotion of the US Defense Advanced Research Program Agency‘s National Cyber Range Project [5].

In this work, we propose leveraging hardware prowess to improve speed performance and emulating the actual system at scale to reveal the finer details in order to enhance modelling and development. Using Field Programmable Gate Array (FPGA), various hardware algorithms and functional modules were designed and implemented on a single chip. This chip can be configured as a processing element in a tightly-coupled multiprocessor or as an independent node in a distributed system. In either scenario, this single chip can be used for emulation and actual computation. With no processor executing codes, drawbacks of inefficient coding and ineffective compiler are avoided.

Section 2 describes the CI hardware agent approach and the experimental platforms. Prototypes of computation modules at various levels of abstraction are presented in Section 3 with analysis. A multiplexer approach to reconfigure hardware on FPGAs is presented and discussed in Section 4. To take advantages of the state-of-the-art reconfigurable FPGAs, the first two stages of the Principal Component Analysis were implemented in both static and dynamic reconfiguration, as presented in the same section. Conclusions drawn from our design and experience are discussed and future works are proposed in Section 5.

2 Hardware CI Agent

Ideally, the hardware CI agent should be housed on a single chip. There are many reasons why this is preferrable and beneficial. First, a single chip can easily fit into an embedded device such as a mobile phone which in many applications act as an independent CI agent. Multiple chipset simply is not an option in many handheld devices due to their small footprint. Second, a single-chip agent can be configured as a co-processor or hardware accelerator for existing computing systems in a distributed, collaborative environment. Third, a single chip can easily be integrated as a node into a tightly-coupled multiprocessor system which can be used either as an emulator or a functional computation engine.

To implement a CI agent in single-chip hardware, there are several options including microcontrollers, Application Specific Integrated Circuits (ASICs), and FPGAs. A microcontroller is a self-contained system typically consisting of a processor, memory, and input-output channels. However, the microcontroller needs to be programmed and each instruction is processed in a regular instruction cycle, rendering it to be a software-driven device. ASIC, as its name implies, is customised for specific applications. Once designed and fabricated, it cannot be reprogrammed. Due to its non-recurring but high design and development cost, and coupled with its inflexibility in reuse, an ASIC is not a suitable platform for a CI agent.

An FPGA is ideal for prototyping and deploying an application. For typical CI application environments, it is expected that the production volume is low, which gives FPGA an edge over the other hardware alternatives. Moreover, the reconfigurability of FPGA during application execution is a big advantage in operating of a CI agent flexibly. Therefore, FPGA is chosen as our prototyping and development platform in the implementation of a hardware CI agent.

2.1 Hierarchical Design

In order to lay the ground work enabling further experiments for complex and high-level concept design, a hierarchical approach is used. Higher level functions utilise lower level operators in a hierarchy as shown in Figure 1. This is akin to a complex design at the system level (which is our eventual goal) using a platform-based design approach incorporating lower level components that can be abstracted and reused [4]. Similarly, the practice of information hiding and reuse is adhered in software implementations. This parallel between software and hardware implementations of the same operators allows us to make sound performance comparison, and to build a library of components in both hardware and software for further experimentation and implementation.

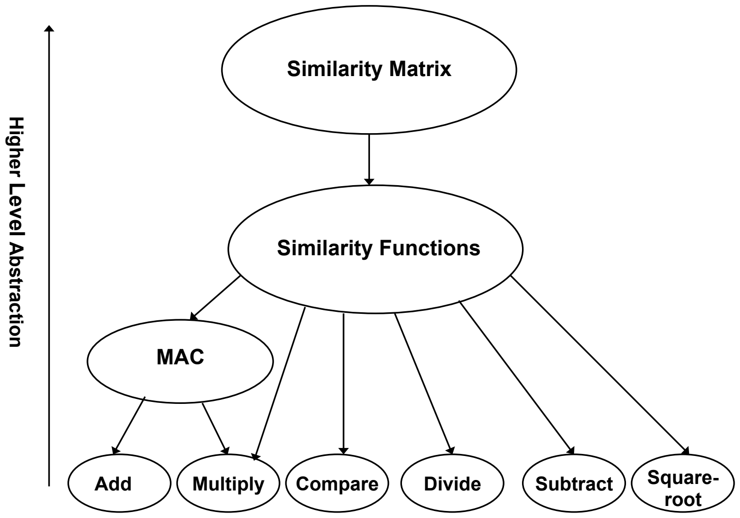


Fig. 1. A Hierarchical Platform-Based Design Approach

2.2 Experimental Platform

All hardware and software experiments were performed on the AMRIX AP-1000 development platform [2], which supports the Xilinx Virtex-II Pro FPGA [27] running at 80 MHz. It has a large FPGA gate capacity and two embedded PowerPC 405 for development purpose. Hardware modules designed in VHDL, incorporating the IEEE Standard Logic Library 1164 [8], were executed on the FPGA to verify their correctness and performance. The Xilinx Integrated Software Environment (ISE) 7.1i [24] was used for hardware development. Modelsim SE and Xilinx Chipscope Pro 7.1i were used to verify the results and functionality of the designs as well as to measure the hardware execution time.

Software modules were written in C and were profiled on a 32-bit RISC MicroBlaze soft core executed on the same FPGA, to ensure fair performance comparisons. The Xilinx Embedded Development Kit (EDK) 7.1i [26] was used to profile software modules and their behaviour, for the purpose of verification.

For all the work presented, no hardware optimization was attempted, and software modules on the MicroBlaze soft core were compiled with no optimisation and with level II optimisation. Level II was selected over other levels because it is the standard optimisation level used for program deployment and it activates nearly all optimisations that do not involve a speed-space tradeoff

[26]. Therefore, the executable would not increase in size in a way that would impact performance because of the limited real estate on chip. Apart from the MicroBlaze soft processor, additional experiments were performed for the software versions using a baseline UltraSparc IIe processor, in order to check the validity of the designs as well as to compare performance among different platforms. CPU execution times were measured using the clock function in C.

3 Illustrative Designs at Various Levels of Abstraction

One important aspect of a CI agent is its capability to function, either as an emulation or a computation engine, at different levels of complexity. If one can examine and profile in details the operations at lower levels, then the hardware agent can be used effectively as a collaborative node in a CI emulation environment. On the other hand, the power of computing at higher conceptual level makes the hardware CI agent an ideal standalone module or an efficient processing element of a processor array in a practical setting.

3.1 Low Level Hardware Operators

As shown in Figure 1, operations such as add, multiple, and divide, are considered to be at the lowest level of the platform-based design hierarchy. These fundamental operators are used by higher level computation modules including multiple-and-accumulate (MAC) and similarity measures. The adder, multiplier and divider were designed and implemented on the FPGA. Their execution times are compared to that of the MicroBlaze software as shown in Table 1.

Table 1. Fundamental Operators Performance

Operator	Hardware (ns)	Software (ns)	Speedup
Adder	12.5	225.0	18
Multiplier	12.5	250.0	20
Divider	12.5	587.5	47

The most important aspect in the design and implementation of low level hardware modules is the ability to profile and monitor their detailed signal propagation and timing in the nanosecond range. This is essential for the investigation of large-scale CI system modelling by emulation.

Multiply-and-Accumulate MAC. The MAC hardware operator was designed as a sequence of multiplier, adder, and an accumulator register that provides a feedback loop to the adder. The multiplier and adder presented earlier were reused. Figure 2 shows the hardware design of MAC which takes advantage of a pipelined configuration with feedback to enhance performance. After the

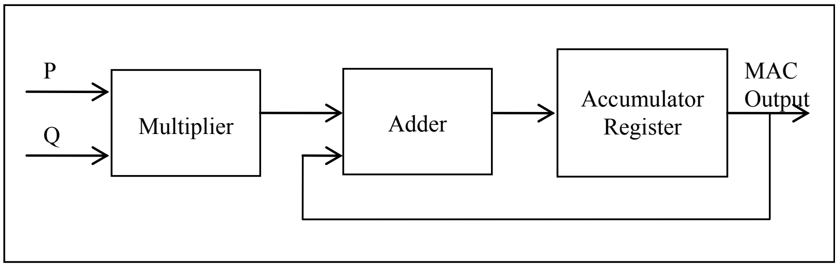


Fig. 2. Multiple-and-Accumulate in a Pipelined Configuration

initial setup time of 2 clock cycles, consecutive results are produced in every clock cycle.

In one experiment, the dot product of two vectors of length 8 was examined. The software version of multiply-and-accumulate used a typical For Loop implementation. The execution time in hardware was 112.5 ns while the equivalent software took 10418 ns, giving an effective speedup of 92.6. With vector lengths in the thousands for large-scale data analysis, the speed performance of the hardware is insurmountable. This is an important illustration of how some commonly used software constructs can be implemented in hardware more efficiently. In this case, a For Loop can be replaced by a pipeline in hardware thus enhancing performance.

3.2 Functional Modules: Similarity Measures

Similarity function is an often used distance measure to compare the similarity among a set of feature vectors. The hardware implementation of a similarity measure illustrates, first, the flexibility of platform-based design; second, the versatility of FPGAs; and finally, a functional module that can be used in various applications which has its counterpart in many software libraries including MATLAB's statistics toolbox [12]. The hardware design of the Extended Jaccard Measure is illustrated in Figure 3. This hardware design takes advantage of this function's inherent parallelism:

$$ExtendedJaccard(p,q) = \frac{\sum_{i=1}^{n} p_i q_i}{\sum_{i=1}^{n} p_i^2 + \sum_{i=1}^{n} q_i^2 + \sum_{i=1}^{n} p_i q_i} \qquad (1)$$

The three dot products have no dependency among them and therefore can be processed in parallel, a feat that a uniprocessor system cannot imitate. The remaining stages of add, subtract, and divide are structured as a pipeline. Thus, parallelism is another dimension of performance improvement using hardware, in addition to pipelining. Furthermore, the platform-based design methodology facilitates these designs with reuse as illustrated in Figure 1.

In total, three similarity measures were designed and implemented on FPGA: Cosine Similarity, Asymmetric Measure, and Extended Jaccard. Experiences

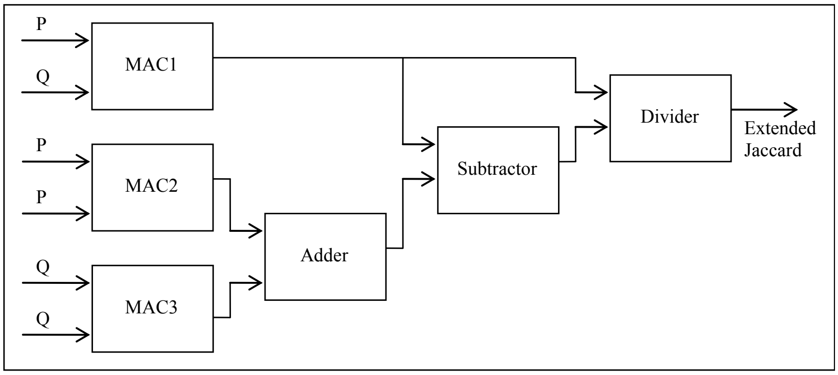


Fig. 3. Extended Jaccard in a Parallel and Pipelined Hardware Configuration

Table 2. Functional Modules Performance

Function	Hardware (ns)	Software (ns)	Speedup
Cosine Similarity	925	33819	35.6
Extended Jaccard	925	33769	36.5
Asymmetric Measure	925	18344	19.8

gained and lessons learned in designing parallel and pipelined hardware are invaluable. Similar to the study of the MAC operator, these three hardware functional modules computed the similarity between two vectors of eight elements. Table 2 compares the hardware and equivalent software performance of the three measures. Due to the synchronous design of the pipelined and parallel modules, all three similarity measure modules take the same time to process.

3.3 Complex Data Structure and Parallelism

Similarity matrix is a commonly used data structure representing the similarity among a set of n vectors. For instance, in the clustering process of n documents, an n X n symmetric matrix with unit diagonal is constructed, with each element of the matrix showing the proximity or similarity of the corresponding row and column documents. A tightly-coupled processor array organized in an SIMD (Single Instruction Multiple Data) [6] fashion is ideal for such data parallelism. As shown in Figure 4, p independent processing elements (PEs) compute the similarity measure between two vectors, as dictated by the control unit. Since the similarity matrix is symmetric with unit diagonal, only the elements of the upper triangle need to be computed. An efficient computation assignment algorithm for the PEs can be found in [17].

In this set of experiments, processor arrays for all three similarity measures were designed and implemented. The number of PEs was varied to illustrate

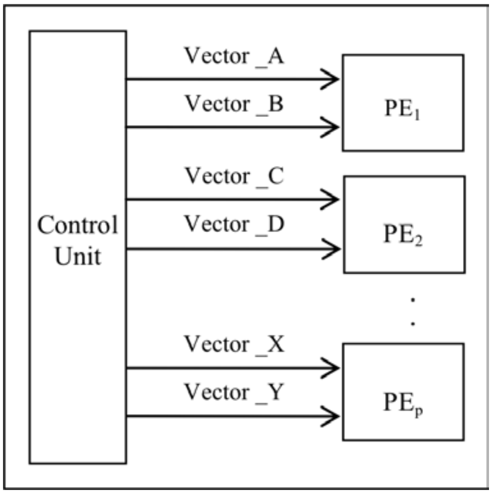


Fig. 4. Processor Array Architecture

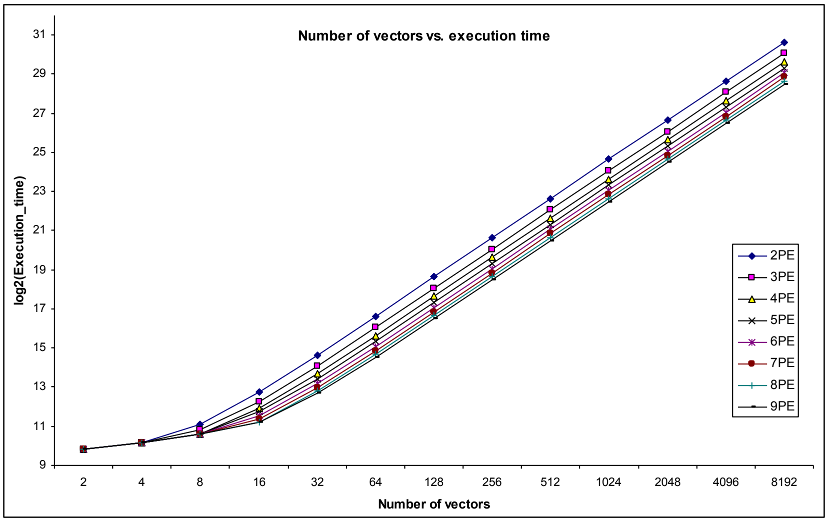


Fig. 5. Processor Array Performance

the flexibility and reconfigurability of the FPGA hardware and the computation assignment algorithm. The number of vectors, with eight elements each, ranged from 2 to 8192. The Cosine Similarity computation times of varying number of PEs and vectors are plotted logarithmically in Figure 5.

The linearity of the computation times shows that the processor array is highly scalable in processing large volume of data. Similar results were obtained with the Asymmetric Measure and Extended Jaccard in a processor array configuration. Even though the incremental gain is monotonically decreasing as the number of PEs increases as shown in Figure 5, the effective speedup is still highly desirable. The results are indeed encouraging as the versatility, scalability, and flexibility of the FPGA-based CI agent can be utilized in various settings. However, the number of PEs or circuitry that can be implemented on a single chip is still limited. FPGA-based reconfigurable computing that alleviates this space limitation problem is discussed next.

4 Reconfigurable CI Agent on FPGAs

Due to the small footprint in many collaboration-enabled mobile devices, it is desirable to implement the hardware CI agent on a single chip. In order to efficiently and effectively utilise the limited space on chip, two design approaches were investigated. The first is to use multiplexer to select the appropriate data path in a design that encapsulates multiple functional modules with extensive reuse of hardware circuitry. The second is to take the state-of-the-art approach of reconfiguring the FPGA chip dynamically.

4.1 Multiplexer-Based Reconfiguration

Extending the designs from Section 3, a single PE consisting of all the modules required to process the three similarity measures has been designed and implemented. Extensive reuse of common modules among the similarity measures enables a PE to be functioned as one of the similarity measures at any given time by selecting and enabling the appropriate submodules with multiplexers as shown in Figure 6.

Space overhead, in this case occupied slices or logic blocks on the FPGA chip, is an important factor to determine the feasibility and effectiveness of the multiplexer-based approach. Figure 7 shows the cost of space for each similarity measure alone, and the combined version of all three. The space saving using the multiplexer-based reconfiguration approach is 63%.

For further investigation, a processing array of eight PEs has been designed and implemented. The PEs can be selected and configured to perform different similarity measures simultaneously. A typical scenario in computing the similarity among a set of objects is to determine what operation to perform next, based on the distribution of the computed similarity. This multiplexer-based processing array architecture gives the ability to process the data using different similarity measures simultaneously. If the data are distinguishable, then one may proceed to the clustering stage. Otherwise, it is desirable to repeat the similarity computation using different algorithms other than the ones just used. This can be accomplished by using dynamically reconfigurable hardware introduced in the next section, and is one of the greatest assets of a CI agent in many decision-making situations in collective intelligence processing.

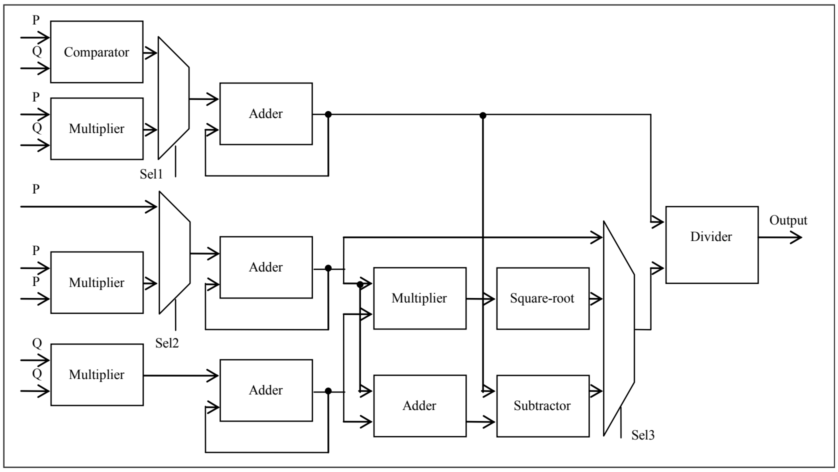


Fig. 6. Multiplexer-based PE for Similarity Measures

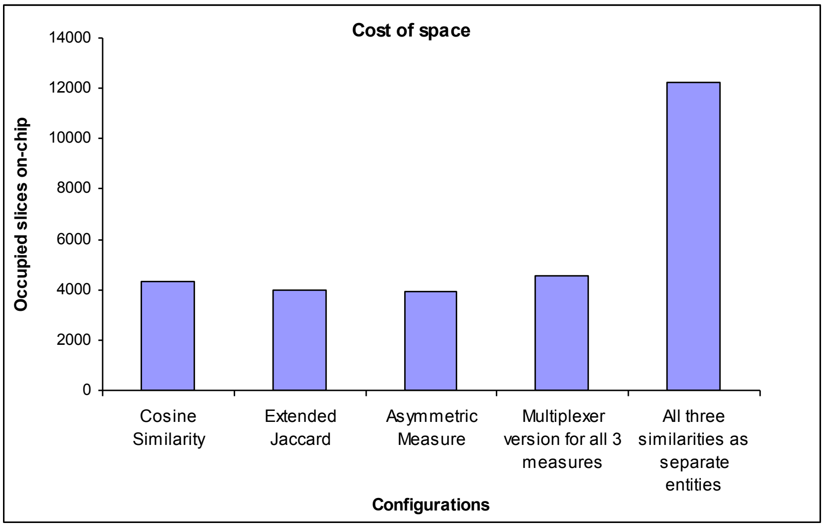


Fig. 7. Space Comparison for Similarity Measure Configurations

4.2 Dynamic Reconfiguration

In previous sections, fundamental operators, functional modules, and processor arrays are shown to be feasible and efficient in FPGA-based hardware implementation. For more complex and high-level concept operations, it might not be possible to fit the corresponding computation circuitry onto a single chip. In addition to multicore architecture, which is essentially a software approach, one can utilise reconfigurable FPGA hardware.

Data analysis is essential in many application domains as it can sieve through large volumes of data to discover useful patterns and valuable knowledge. Many of the data analysis algorithms require complex computations thus processing speed is a primary concern. For instance, Principal Component Analysis (PCA) [15], used extensively in preliminary data analysis and dimensionality reduction, consists of several stages of computation. PCA is used as a case study to illustrate how FPGA-based reconfigurable hardware can be used in complex and compute intensive data analysis applications found in the realm of collective intelligence.

4.3 Design Approach and Development Platform

Both software and static/dynamic reconfigurable hardware (denoted as SRH and DRH) used the same hierarchical platform-based design approach introduced in Section 2.1. All software and hardware designs were experimented on the Xilinx ML605 FPGA development board [25], which provides dynamic reconfiguration.

There are two ways to reconfigure Xilinx Virtex-6 FPGAs: multiboot and partial reconfiguration. Multiboot is typically carried out by an on-chip controller, using reconfiguration files called bitstreams, stored in an external memory. Without the need to have an external controller to manage reconfiguration, self reconfiguration is possible by implementing the on-chip controller as a state machine for quick decision making. However, the entire chip must be reconfigured and loading bitstreams from a memory device external to the chip incurs high overhead.

Partial reconfiguration allows the reconfiguration of part of the chip while the other parts are still operational. This has the advantage of hiding the reconfiguration latency while computation is still in progress. Partial bitstreams can be stored either on chip or in external memory. Similar to multiboot, the on-chip controller can act independently and make quick decisions.

Both multiboot and partial reconfiguration are ideal for implementing self-learning and adaptive agent. In the following experiments, partial reconfiguration was used due to its property of being able to compute and reconfigure at the same time. This enhances speed performance and facilitates decision making.

4.4 Principal Component Analysis Case Study

The various stages of PCA have been designed, implemented and studied separately. To illustrate how dynamic reconfiguration works and benefits complex procedures, only the first two stages of PCA, Mean and Covariance Matrix computations, were used for simplicity. The original equations to generate the Mean and the Covariance Matrix are [10]:

$$\overline{X}_j = \frac{\sum_{i=1}^{n} X_{ji}}{n} \qquad (2)$$

$$CM(X_j, X_{j+1}) = \frac{\sum_{i=1}^{n}(X_{ji} - \overline{X}_j)(X_{(j+1)i} - \overline{X}_{j+1})}{n-1} \tag{3}$$

These equations are modified in order to streamline the hardware implementation:

$$\overline{X}_j = \sum_{i=1}^{n} X_{ji} \tag{4}$$

$$CM(X_j, X_{j+1}) = \frac{\sum_{i=1}^{n}(X_{ji} - \overline{X}_j)(X_{(j+1)i} - \overline{X}_{j+1}}{n(n-1)} \tag{5}$$

These slight modifications allow the delay of the Mean's division to be computed at the output stage of the Covariance Matrix, thus enabling the use of integer operations for both modules. Again, this illustrates the flexibility of implementing algorithms in hardware. The hardware module for the Mean computation utilises a 2-stage adder and accumulator configuration with a feedback loop is shown in Figure 8, which also forms part of the Covariance Matrix hardware as shown in Figure 9.

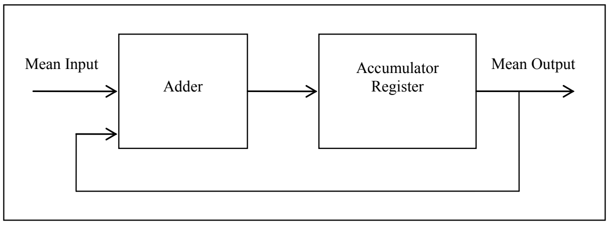


Fig. 8. Mean Computation Using Feedback Loop

The static reconfigurable hardware module (SRH) consists of both the Mean and the Covariance Matrix on the same chip. The dynamic reconfigurable version (DRH) has the Mean module downloaded to the FPGA first, and after its computation is completed, the Covariance Matrix module is then downloaded and computed.

Space Analysis. In the static configuration, the number of on-chip slices for the Mean and the Covariance Matrix modules are 2579 and 3434 respectively, giving a total of 6031. For the dynamic reconfiguration, the Covariance Matrix module occupies more space than the Mean module, hence the maximum number of slices required on chip as per the Covariance Matrix module is 4279. This constitutes a space saving of 30%. The space overhead required in dynamic reconfiguration is about 538 slices, less than 10% of the static reconfiguration case.

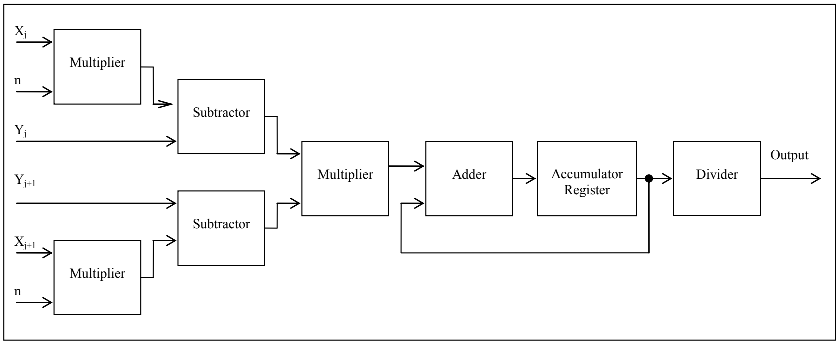


Fig. 9. Covariance Matrix Computation in a 6-Stage Pipeline

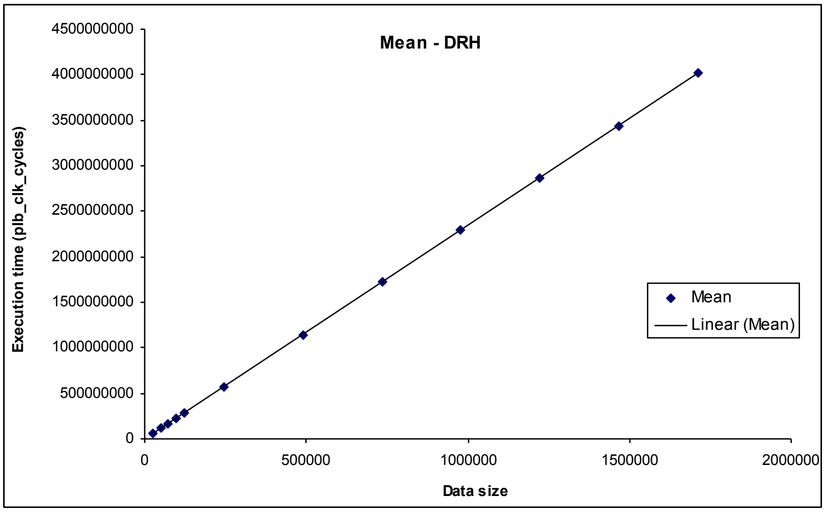


Fig. 10. Mean Computation with Varying Data Size in Dynamic Reconfiguration Case

Time Analysis. To further understand the execution time for both static and dynamic reconfigurations, experiments were carried out using a benchmark data set. The Optical Recognition of Handwritten Digits Data Set [1] used, has 3823 64-element vectors, giving a data size of 244672. Both reconfiguration hardware modules, SRH and DRH, were designed with variable inputs including the number of vectors, the number of elements per vector, and hence the data size.

Figure 10 shows the execution time in clock cycles with varying data size for the Mean computation in the dynamic reconfiguration case. The linear behaviour is the same for the Covariance Matrix computation in dynamic reconfiguration, and both the Mean and Covariance Matrix in static reconfiguration. This further affirms the linearity of the hardware computation, hence, the scalability of the hardware algorithms. The dynamic reconfiguration time from Mean to Covariance Matrix took 515 ms, which is insignificant with increasing data size.

The speed performance of static and dynamic configurations with varying data size is comparable and similar. The reconfiguration overhead for the dynamic case is insignificant except when the data size is small. This confirms that time penalty is insignificant in dynamic reconfiguration. More results and analysis of this case study can be found in [18].

5 Conclusion and Future Work

A hardware collective intelligence agent is presented with various design and implementation methodologies on FPGA. Indeed, all hardware designs are superior than their software counterparts in speed performance. The dynamic reconfigurability is of particular interest as it can accommodate complex computations without sacrificing speed performance. In addition, the single-chip philosophy works well with a distributed, collaborative environment, either as a stand-alone CI engine, or an intelligent node in a sensor network, or part of a multiprocessor. Moreover, the flexibility, versatility, and scalability offered by the FPGA hardware enable large-scale emulation to study CI models as well as actual deployment. Handheld embedded devices having small footprint and requiring processing power can also take advantage of a small piece of FPGA chip, for real-time applications such as fingerprint and iris identification. Depending on its capacity, an FPGA chip in early 2010s typically costs from US$100 upwards to $3000, making it a reasonably cost-effective device. Currently, other high-level computation modules including clustering and classification algorithms are being investigated for implementation on dynamic reconfigurable FPGAs.

References

1. Alpaydin, E., Kaynak, C.: Optical Recognition of Handwritten Digits Data Set. UCI Machine Learning Repository, University of California, School of Information and Computer Science (1998)
2. Amirix, Inc., http://www.amirix.com/products
3. Baumoel, U., Georgi, S., Ickler, H., Jung, R.: Design of new business models for service integrators by creating information-drien value webs based on customers collective intelligence. In: International Conference on System Sciences, pp. 1–10 (2009)
4. Densmore, D., Passerone, R.: A Platform-Based Taxonomy for ESL Design. IEEE Design and Test of Computers 23(5), 359–374 (2006)
5. Defense Advanced Research Projects Agency (DARPA): Broad agency announcement: national cyber range (2008), https://www.fbo.gov/index?s=opportunity&mode=form&tab=core&id=16ce874dacb9910e7327e7545a054df8
6. Flynn, M.: Some computer organizations and their effectiveness. IEEE Transactions on Computers C-21, 948 (1966)

7. Gregg, D.: Designing for collective intelligence. Communications of the ACM 53(4), 134–138 (2010)
8. IEEE Standard Multivalue Logic System for VHDL Model Interoperability 1164-1993, `ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=2823`
9. Iguider, Y., Morita, H.: A collective intelligence based business-matching and recommending system for next generation e-marketplaces. In: IEEE Symposium on Computers and Informatics, pp. 489–494 (2011)
10. Jackson, J.: A User's Guide to Principal Components. Wiley Interscience (2003)
11. Lee, D., Kim, J., Lee, H.: Collective intelligence based collaborative learning platform. In: International Conference on Information and Communication Technology Convergence, pp. 553–554 (2010)
12. MathWorks: MATLAB Statistics Toolbox 7.4 (2011)
13. Narasimhan, S., Paul, S., Bhunia, S.: Collective computing based on swarm intelligence. In: ACM/IEEE Design Automation Conference, pp. 349–350 (2008)
14. Neville, S., Li, K.: The rational for developing larger-scale 1000+ machine emulation-based research test beds. In: International Conference on Advanced Information Networking and Applications, pp. 1092–1099 (2009)
15. Pearson, K.: On lines and planes of closest fit to systems of points in space. Philosophical Magazine 2(6), 559–572 (1901)
16. Pentland, A.: Collective intelligence. IEEE Computational Intelligence, 9–12 (2006)
17. Perera, D., Li, K.: Parallel Computation of Similarity Measures Using an FPGA-Based Processor Array. In: IEEE International Conference on Advanced Information Networking and Applications, pp. 955–962 (2008)
18. Perera, D., Li, K.: FPGA-based reconfigurable hardware for compute intensive data mining applications. In: International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing, pp. 100–108 (2011)
19. Rasmussen, S., et al.: Collective intelligence for decision support in very large stakeholder networks: the future US energy system. In: IEEE Symposium on Artificial Life, pp. 468–474 (2007)
20. Rojas, M., Mesa, J.: Collective knowledge of the Web: source of information of process of business intelligence. Colombian Computing Congress, 1–6 (2011)
21. Singh, V., Gupta, A.: Agent based models of social systems and collective intelligence. International Conference on Intelligent Agent and Multi-Agent Systems, 50–56 (2009)
22. Vergados, D., Lykourentzou, I., Kapetanios, E.: A resource allocation framework for collective intelligence system engineering. In: International Conference on Management of Emergent Digital EcoSystems, pp. 182–188 (2010)
23. Vivacqua, A., Borges, M.: Collective intelligence for the design of emergency response. In: International Conference on Computer Supported Cooperative Work in Design, pp. 623–628 (2010)
24. Xilinx, Inc., ISE Design Suite (2010), `www.xilinx.com/support/documentation/dt_ise.htm`
25. Xilinx, Inc., ML605 Hardware User Guide (2011), `www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf`
26. Xilinx, Inc., Platform Studio User Guide: Embedded Development Kit EDK 7.1i (2005), `www.xilinx.com/tools/platform.htm`
27. Xilinx, Inc., Virtex-II Pro FPGA User Guide (2007), `www.xilinx.com/support/documentation/user_guides/ug012.pdf`
28. Xilinx, Inc., Virtex-6 FPGA Configuration User Guide (2010), `www.xilinx.com/support/documentation/user_guides/ug360.pdf`

SimISpace2: A Simulation Platform for Exploring Strategic Knowledge Management Processes

Martin Ihrig

The Wharton School, University of Pennsylvania, Philadelphia, PA 19146, USA
ihrig@wharton.upenn.edu

Abstract. *SimISpace2* is an agent-based simulation environment designed to simulate strategic knowledge management processes, in particular knowledge flows and knowledge-based agent interactions. It serves as a general knowledge management engine that, through a user-friendly graphical interface, can be adapted to a wide range of knowledge-related applications. Its purpose is to improve our understanding of how knowledge is generated, diffused, internalized and managed by individuals and organizations, under both collaborative and competitive learning conditions.

Keywords: agent-based simulation, strategic knowledge management, I-Space, knowledge development and diffusion.

1 Introduction

SimISpace2 is an agent-based graphical simulation environment designed to simulate strategic knowledge management processes, in particular knowledge flows and knowledge-based agent interactions. Its conceptual foundation is provided by Boisot's [1,2] work on the *I-Space*. The predecessor version of *SimISpace2* [3] had been used to study intellectual property rights policies [4], to explore the spatial dimension of knowledge flows [5], and to configure simulations that model the knowledge-transfer dilemma facing the counter-terrorism community [6]. The simulation environment presented in this paper, *SimISpace2*, is a total redevelopment (both conceptual and technical) which has been built from scratch using the programming language $C^\sharp$ [7]. It is noticeably different from existing approaches for modeling the physical world, since it makes it possible to "model the multiple knowledge-specific activities required of the knowledge lifecycle" [5,6]. Users can study the effects of individual strategic knowledge management actions and explore knowledge processes at the macro level. *SimISpace2* is one of the very few simulation environments that enables researchers to model distinct knowledge assets and to analyze micro and macro effects of knowledge-based strategies of interacting agents. It thereby presents itself as a unique tool for conducting innovative social science and management research on the dynamic evolution of knowledge. It can also be used in the context of intelligent large-scale systems and to study complex adaptive systems of knowledge and social networks.

The structure of this paper is as follows. First, we outline what the user can expect from the simulation and explain the underlying concepts of the *SimISpace2* environment, including how the properties of the basic entities - agents and knowledge assets - are determined. Second, we describe in detail how knowledge assets are represented in the simulation model and what properties they can have, as well as discussing the importance of knowledge networks, the difference between owning and possessing knowledge, and the value of knowledge. Third, we depict the different properties agents can have and give a detailed description of the various actions agents can perform. Fourth, we conduct a couple of simple simulation experiments to ensure the software fully implements and represents *I-Space* theory.

2 Fundamentals

2.1 A Brief Description of the Innovative Simulation Environment

SimISpace2 is an agent-based simulation environment [8] that is being developed to implement the main features of the *I-Space* [1,2] as a conceptual framework. The Information Space, or *I-Space*, is a conceptual framework that has been used to study knowledge flows in diverse populations of agents - individuals, groups, firms, industries, alliances, governments, and nations. Building on this, *SimISpace2* is designed to serve as a general knowledge management engine that, through a user-friendly graphical interface, can be adapted to a wide range of knowledge-related applications. Its purpose is to improve our understanding of how knowledge is generated, diffused, internalized and managed by individuals and organizations, under both collaborative and competitive learning conditions.

We start by creating domain-specific knowledge networks that agents are required to discover and exploit by investing either in individual learning processes - codification, abstraction, etc. - or in interactions with other agents. The two together make up a social learning process. *SimISpace2* allows a user to specify different knowledge groups, agent groups, discovery conditions, knowledge spillover conditions, cost conditions and reward conditions.

Knowledge assets are fundamentally different from physical assets. *SimISpace2* has been created to account for this. The *I-Space* relates the degree of *structure of knowledge* (i.e. its level of codification and abstraction) to its *value* and *diffusibility* as that knowledge develops. This complex relationship has been incorporated in the architecture of the simulation environment. In addition, the simulation has two features that capture two other critical properties of knowledge assets, which distinguish them from physical assets. First, knowledge items are stored separately from agents who own them. Unlike physical assets, knowledge assets can be owned by more than one agent. For example, an agent can license or trade its knowledge, but still retains knowledge of that item. Second, as agents develop their specific items of knowledge, they retain all the prior knowledge associated with the newly developed knowledge. Thereby, the agent can at any time draw on all the past forms of the knowledge item to exploit it.

Complex scenarios can be modeled with *SimISpace2* by assigning distributions for each of the initial properties of agent and knowledge groups. More than 170 direct parameters, both general and action-specific, can be set for each of the agent and knowledge groups created. During the simulation runs, agents compete and cooperate in performing three different kinds of actions:

– Managing knowledge (e.g. discovering, codifying & abstracting, protecting, exploiting, disposing, etc.)
– Exchanging knowledge (e.g. trading, licensing)
– Networking and being in motion (e.g. meeting, relocating, relaxing, exiting & entering).

The above actions and interactions form the basis for the emergent properties of agents (e.g. stock of knowledge, financial and experience funds, location, etc.) and of knowledge assets (e.g. diffusion, location, structure, obsolescence, etc.). *SimISpace2* has the following innovative features that make it different from other simulation programs:

a) It models information flows by implementing a theoretical framework that is novel, powerful, and growing in use.
b) It is the only agent-based simulation of knowledge flows that is based on a comprehensive theory of agent-based knowledge evolution (the *I-Space* theory).
c) It can track and analyze multiple collectivities' social learning processes, their discovery strategies, and the agent interaction strategies.
d) By analyzing the complex networks of agent interactions, it can track the emergence of institutional structures and gain an understanding of the reciprocal influence of process and structure in the knowledge domain.

2.2 The Big Picture

There are many conditions, properties, and tools integrated into the *SimISpace2* simulation environment. However, before examining each of the individual features, it is necessary to realize the fundamentals that underlie the simulation's intricate composition. In its current version, the basic purpose of *SimISpace2* is to explore how user-defined agents evolve as measured by knowledge discovery and growth of financial funds.

There are basically three sources in the simulation that an agent draws on to obtain or develop new knowledge. The first source is a global pool of knowledge that an agent accesses and extracts pieces of knowledge from. The second source of knowledge is from the knowledge portfolio that an agent already possesses, which can be internally developed into new knowledge through structuring and learning processes. Finally, an agent can obtain knowledge from other agents through certain actions such as scanning or trading. How different agents utilize these three sources ultimately depends on the complex attributes that the user assigns to each agent.

Knowledge discovery affects the other principal indicator of an agent's development, its financial funds. Agents capitalize on knowledge discovery, and they influence their financial funds by executing several actions such as trading, licensing, and exploiting. For example, when an agent sells or licenses knowledge to another agent, its financial funds increase. Similarly, an agent's financial funds enlarge when it exploits its knowledge. As will be shown, there are other costs and factors that affect an agent's financial funds, but these actions are among the main determinants. Financial funds are important to understanding an outcome of a simulation model, as they measure the growth of agents and thereby present opportunity for analysis.

As this simulation environment's extensive components are individually explained, it is important to consider them in the larger framework. They should be appreciated in the context of how they collectively contribute to the evolution of knowledge discovery and financial funds. This paper seeks to answer the following questions:

1. What can the user expect from the simulation?
2. What are the properties of the simulation that the researcher can use to build an application-specific model?
3. What happens during the simulation runs?
4. How do we know the simulation works?

Each of these questions is addressed in different sections of the paper (Fig. 1). Due to the complexity of the software program and the different processes it models, it is very difficult to describe all the features in a purely linear fashion. Therefore, we suggest you read through this paper fairly quickly to get the main gist and then reread for the details and connections. Since this is a semi-technical description, it is hard to convey all the complex relationships at once. Nevertheless, they are all listed and described, so that researchers can get a full understanding of the mechanisms.

2.3 Getting Started: Specifying Different Knowledge and Agent Groups

Two major forms of entities can be modeled with *SimISpace2*: agents and knowledge items/assets. When setting up the simulation, the user defines agent groups and knowledge groups with distinct properties. Individually definable distributions can be assigned to each property of each group. When the simulation runs, the individual group members (agents and knowledge items) are assigned characteristics in accordance with the distribution specified for the corresponding property for the group of which they are a member. For a particular agent group for example, the property that determines the starting level of the financial funds of agents can be set to have a uniform distribution with lower bound 100 and upper bound 150. Individual agents in that group will then have their initial financial funds set to between those limits when they are created.

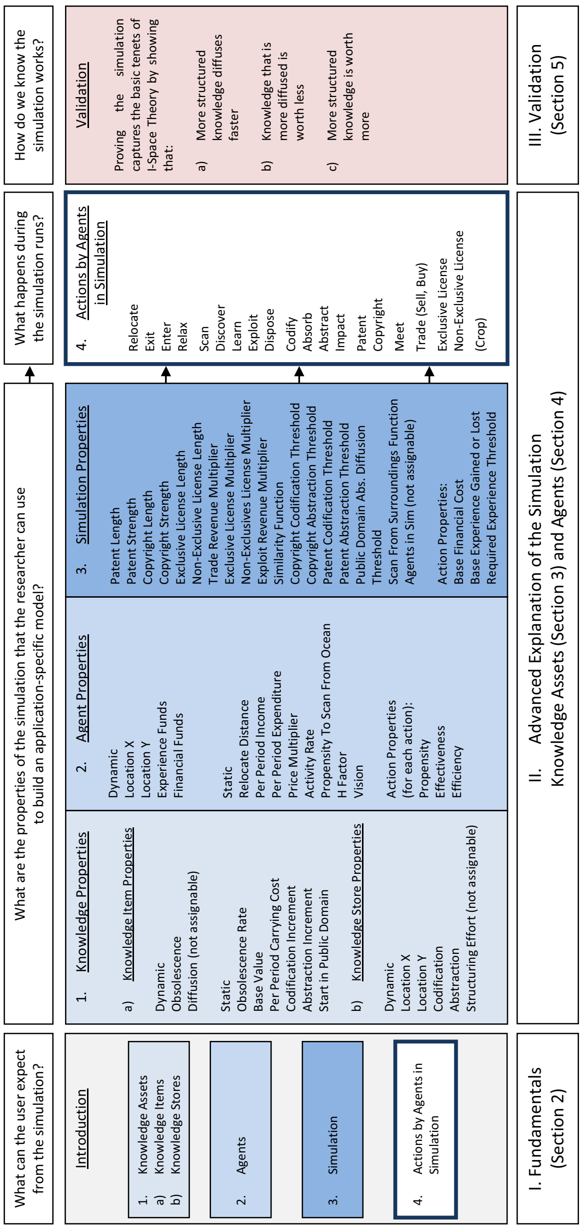

What can the user expect from the simulation?
What are the properties of the simulation that the researcher can use to build an application-specific model?
What happens during the simulation runs?
How do we know the simulation works?
Introduction
1. Knowledge Assets
a) Knowledge Items
b) Knowledge Stores
2. Agents
3. Simulation
4. Actions by Agents in Simulation
1. Knowledge Properties
a) Knowledge Item Properties
Dynamic
Obsolescence
Diffusion (not assignable)
Static
Obsolescence Rate
Base Value
Per Period Carrying Cost
Codification Increment
Abstraction Increment
Start in Public Domain
b) Knowledge Store Properties
Dynamic
Location X
Location Y
Codification
Abstraction
Structuring Effort (not assignable)
2. Agent Properties
Dynamic
Location X
Location Y
Experience Funds
Financial Funds
Static
Relocate Distance
Per Period Income
Per Period Expenditure
Price Multiplier
Activity Rate
Propensity To Scan From Ocean
H Factor
Vision
Action Properties (for each action):
Propensity
Effectiveness
Efficiency
3. Simulation Properties
Patent Length
Patent Strength
Copyright Length
Copyright Strength
Exclusive License Length
Non-Exclusive License Length
Trade Revenue Multiplier
Exclusive License Multiplier
Non-Exclusives License Multiplier
Exploit Revenue Multiplier
Similarity Function
Copyright Codification Threshold
Copyright Abstraction Threshold
Patent Codification Threshold
Patent Abstraction Threshold
Public Domain Abs. Diffusion Threshold
Scan From Surroundings Function
Agents in Sim (not assignable)
Action Properties:
Base Financial Cost
Base Experience Gained or Lost
Required Experience Threshold
4. Actions by Agents in Simulation
Relocate
Exit
Enter
Relax
Scan
Discover
Learn
Exploit
Dispose
Codify
Absorb
Abstract
Impact
Patent
Copyright
Meet
Trade (Sell, Buy)
Exclusive License
Non-Exclusive License
(Crop)
Validation
Proving the simulation captures the basic tenets of I-Space Theory by showing that:
a) More structured knowledge diffuses faster
b) Knowledge that is more diffused is worth less
c) More structured knowledge is worth more
I. Fundamentals (Section 2)
II. Advanced Explanation of the Simulation Knowledge Assets (Section 3) and Agents (Section 4)
III. Validation (Section 5)

Fig. 1. Structure of paper

Agents and knowledge items cannot join or leave groups. They start off being a member of a group from the beginning of the simulation and remain a member until the end of the simulation. Even if they die, they remain associated with their group. The static nature of group membership has been implemented to facilitate easier analysis. If groups had dynamic membership, a consistent analysis of the properties of their members would be harder since the actual members would change from period to period. To prevent overlapping distributions, a participant - an agent or a knowledge item - is only ever a member of a single group. Each group has its own distributions for each property; and if an agent (or knowledge item) were a member of more than one group, then conflicts would arise as to which distribution to base their property values from.

As a result, group size is static, there are no new entrants. The number of members is entirely determined by the start number specified for the group by the user. During the start-of-simulation processing (bootstrapping), the simulation iterates through all the agents and knowledge items and uses random-number generation to assign each agent's and knowledge item's characteristics according to the distribution specified for each property. That way, the agent and knowledge groups contain agents and knowledge items with approximately the distribution specified for each property. For example, each agent will start with a given level of financial funds, according to the distribution specified for its agent group; and each knowledge item will have a particular base value, according to the distribution specified for its knowledge group.

2.4 Distributions

Following are the names and parameters for each type of distribution available. The distributions are a subset of the distributions provided by Crystal Ball (see Hillier, Hillier and Hillier [9] for further information on Crystal Ball's distributions). The researcher can chose from five different distribution types to set the properties for agent groups, knowledge groups and simulation-wide settings.

Uniform distribution. Values are evenly distributed between lower and upper limit, which the user can specify. For example, when users want to set the property determining the 'geographic' starting location of agents in an agent group, they can assign a uniform distribution. This would mean that the agents of this agent group are equally spread out between the minimum and the maximum value.

Normal distribution. The normal distribution has a symmetric bell curve and the user can specify the mean and the standard deviation. For example, when researchers want to set the property determining the starting level of financial funds of agents in an agent group, they could assign a normal distribution, reflecting some agents start off with a financial endowment that is average, some with one that is above average, and some with one that is below average.

Triangle distribution. A triangular distribution can be used to approximate an asymmetric bell curve, and it is specified by its minimum, maximum and mean values. The user sets the lower limit, the mode and the upper limit.

Exponential distribution. For the exponential distribution, the user can specify lambda. Here, all participants in the participant-group, be it agent or knowledge, get a value for a particular property assigned that is drawn from an exponential probability distribution.

Constant distribution. The constant distribution uses the mean variable to determine a point distribution rather than a function. It means all participants in the participant-group have the same value for that property. This could technically be modeled using a uniform distributions (with lower limit and upper limit equal to each other), but that would require setting multiple parameters, so we provide the constant distribution for convenience. It makes it easier for example to set up multi-simulations as the user can vary a single parameter in the multi-simulation rather than having to vary multiple parameters in a synchronized fashion.

2.5 Simulation User Processes

Simulation user processes are processes performed by the human user of the *SimISpace2* environment in order to configure simulations. Different windows/interfaces let the user specify the settings (Fig. 2). There are three main steps: create basic entities for a new simulation, manage parameter settings, and run the simulation.

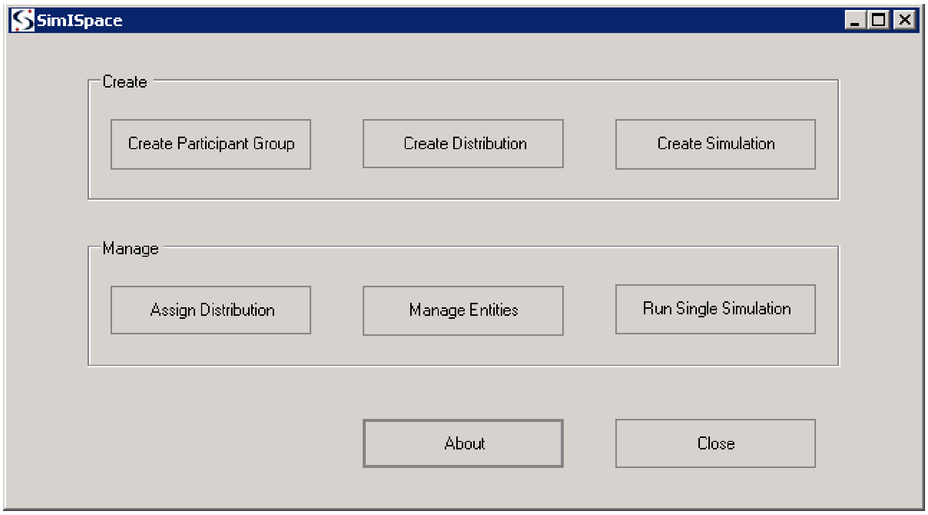


Fig. 2. Screenshot of the *SimISpace2* software main interface

The following tasks have to be completed to set-up and run a simulation:

Create participant group. The first step is to create agent groups and knowledge groups. The user specifies the name of the group as well as the starting number of members in each group. Participant groups are created independently of simulations as they can be used for (assigned to) multiple simulations. The specific properties of the group created by the user are set in a separate step/window (see below).

Create distribution. The user creates distributions by selecting the type of the distribution and specifying the distribution's parameter values. As described above, the parameterized distributions are used to establish the participants' properties.

Create simulation. The user has to configure the basic parameters and properties of the simulation. This involves:

- specifying the participating agent and knowledge groups,
- setting the industry revenue multipliers,
- determining the prior stock of knowledge for agents (knowledge items that agents possess from period 1 on),
- establishing links between knowledge items (intra- and inter-knowledge-group links),
- specifying prior acquaintances, i.e. a network of agents that know each other (that have already met),
- creating DTI knowledge items (see below for description of DTI knowledge). See further down for a detailed description of the different properties.

Assign distribution. Assigning distributions to group properties is implemented as drag-and-drop. The process allows users to link group properties for agent and knowledge groups they have created to distribution instances they have defined when they added distributions. Several general simulation properties that apply to all participant groups are also determined by distributions.

Manage entities. The user can always revisit the settings and edit the parameter values of all participant groups, distributions and simulations that were created.

Run simulation. The user can view the list of simulations in the database and choose one to run. The user has to specify how many times the simulation is supposed to run (1 to N times) and determine how many periods there are per run (1 to K periods).

3 Knowledge Assets

3.1 Terminology

Knowledge in the simulation environment is defined as a 'global proposition'. The basic entities are *knowledge items.* Based on the *knowledge group* they belong to, those knowledge items have certain characteristics. All knowledge items together make up the *knowledge ocean* - a global pool of knowledge. Agents access the knowledge ocean, pick up knowledge items, and deposit them in knowledge stores. A knowledge store is an agent's personal storage place for a knowledge item. Each knowledge store is local to an agent, i.e. possessed by a single agent. Knowledge stores as containers *hold* knowledge items as their contents. This happens after agents obtain a knowledge item. Examples of a knowledge store include books, files, tools, diskettes, and sections of an agent's brain. There is only one knowledge item per knowledge store, i.e. each knowledge item that an agent possesses has its own knowledge store. If an agent gets a new knowledge item (whether directly from the knowledge ocean or from other agents' knowledge stores), a new knowledge store for that item is generated to hold it. The following graph illustrates the distinctions we make (Fig. 3).

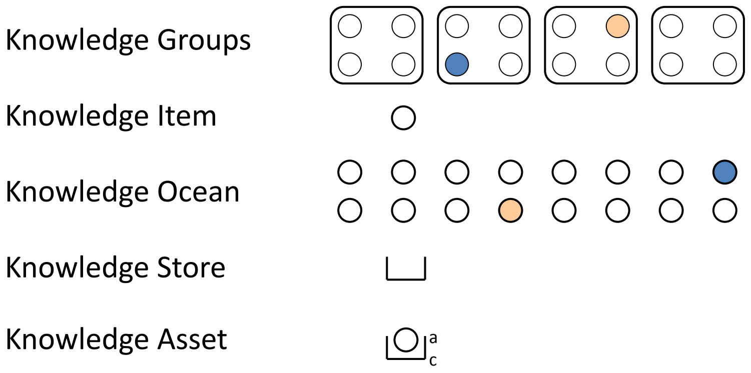


Fig. 3. From knowledge ocean to knowledge asset

3.2 Knowledge Item vs. Knowledge Store

The concept of a knowledge item has been separated from the concept of a knowledge store so that knowledge can be traced and tracked as it evolves. If knowledge items are taken from the same common pool and stored in different agents' knowledge stores, then it is possible to see when two (or more) agents have the same knowledge item. This is very useful for seeing how knowledge diffuses. Contrast this to the case where agents are allowed to arbitrarily create their own knowledge. This would make it very difficult to determine the provenance of knowledge, as it would be very hard to determine when two pieces of knowledge held by different agents are in fact the same knowledge.

The importance of the separation between a global pool of knowledge and local knowledge stores can also be illustrated by looking at abstraction and codification (which applies only to knowledge stores and not to knowledge items). Knowledge stores are held by an agent and held at a particular level of codification and abstraction. This means that if the agent codifies its knowledge and makes it more abstract, the properties of the knowledge item are not changed but the corresponding knowledge store gets a higher degree of codification and abstraction. So the knowledge item is held at a certain level of abstraction and codification in that knowledge store. Knowledge stores are about the *form*, knowledge items about the *content* of knowledge.

When it comes to codification and abstraction, the simulation uses a so-called *string-of-pearls* mechanism. Each time an agent codifies or abstracts knowledge (the knowledge store that holds a knowledge item), a new, i.e. an additional, knowledge store is created with the new level of codification or abstraction. The same applies to the opposite actions to codification and abstraction: absorption and impacting. The knowledge item in those knowledge stores is always the same, only the level of codification and abstraction of the knowledge stores changes as does its 'structuring effort'. Structuring effort is the absolute distance that a knowledge store and its predecessors have moved through the *I-Space* in the codification and abstraction dimensions. It is a means of implementing version of knowledge stores. The string-of-pearls mechanism is one of the ways the cumulative nature of knowledge is modeled.

3.3 Knowledge Properties

Each knowledge item belongs to a particular knowledge group. As already mentioned, at the outset, the properties of those knowledge groups can be set by assigning distributions. The individual knowledge item's characteristics depend on the distributions set on the group level. There are static and dynamic properties. With static properties, the characteristics are assigned to knowledge items at the beginning of the simulation and do not change. With dynamic properties, we look at the characteristics of an individual knowledge item that evolve throughout the simulation. The characteristics can be tracked during the simulation by following the values of each property for a particular individual knowledge item at specific points in time. Dynamic properties have a starting level - determined by the distribution set on the group level - but are recalculated after each period based on occurrences in that period. Characteristics of knowledge stores are influenced by the characteristics of the knowledge items they hold and the actions agents who possess them perform. The following subsections describe the properties of knowledge items and knowledge stores.

Properties of Knowledge Items

Obsolescence. The obsolescence property for each knowledge item is recalculated at the end of each period (dynamic property). Obsolescence at the end of the period is the obsolescence at the start of the period plus the obsolescence rate ($O_n = O_{n-1} + o_r$). Obsolescence varies from zero to one, where one means the knowledge is worthless (i.e. completely obsolete).

Obsolescence rate. The obsolescence rate property for each knowledge item specifies at what rate (units per period, rather than percent per period) that knowledge item becomes obsolete.

Base value. It is the basic value of a knowledge item, which may be different from the price that agents set for that knowledge item when they trade it (see further down for more information on the value of knowledge).

Per period carrying cost. It is the cost of knowing or holding a particular knowledge item, per period. Knowledge with a higher level of codification and abstraction has a lower carrying cost. By assigning a negative value (negative cost = gain), agents can be rewarded to develop knowledge, i.e. obtain more knowledge stores. Knowledge with a higher level of codification and abstraction has a higher carrying gain.

Abstraction increment and codification increment. It is the increment by which abstraction and codification can increase. Note that a knowledge item itself does not have a level of abstraction or codification - only a knowledge store has a level of abstraction and codification. However, knowledge items have an abstraction increment and a codification increment, which help determine how fast the knowledge stores containing them can be abstracted and codified.

Start in public domain. The user can specify whether the knowledge items of a knowledge group are common knowledge or not. If they start in the public domain, agents will not be able to patent or copyright (own) them.

Diffusion. Diffusion is calculated as the percentage of agents who possess a particular piece of knowledge. When calculating the number of agents, those agents who are cropped (i.e. those who died after exhausting their resources) or exited (and not re-entered) are not counted. The diffusion property for each knowledge item is calculated dynamically at the end of each period. As it is an emergent property of the simulation, it cannot be set in advance. (N.B.: Diffusion is also not considered an action of any participant.) Consequently, diffusion rate is not allowed to be stored or set, as rate of diffusion is also an emergent property of the system.

Diffusion of knowledge among agents has two components. First, it is a consequence of voluntary exchange through agent interactions such as trading or licensing. Second, it results from spillovers through scanning from other agents. Those spillovers reflect the degrees of codification and abstraction of knowledge stores (and, by implication, of knowledge items in the stores), because more abstract and codified knowledge is easier to scan. Since scanning is more likely to happen on more codified and abstract knowledge, diffusion will be faster for more codified and abstract knowledge. Thus, patented and copyrighted knowledge, which has had to cross a certain codification and abstraction threshold, is usually also easy to scan (but it is harder to exploit - see further down for details). Diffusion also happens when agents obtain knowledge directly from the knowledge ocean. Here, the links between knowledge items come into play: as will be described below, agents are more likely to scan linked items.

Properties of Knowledge Stores

Location. Knowledge stores have a specific two-dimensional location (X/Y) in the *SimWorld*, the 100 by 100 grid knowledge and agents occupy. Thus, both X and Y can have values of between 0 and 100.

Codification and abstraction. Codification and abstraction are properties of knowledge stores, and not properties of knowledge items. The level of codification and of abstraction ranges between zero and one. Knowledge items are held by multiple agents, and one agent's investment in codification or abstraction does not influence the codification and abstraction level of the same knowledge item held by another agent. Therefore, knowledge items do not have a level of codification and abstraction, only knowledge stores do. Thus, abstraction and codification are local (i.e. particular to a given knowledge store, which resides in some location), as opposed to global. However, the user can set a 'starting degree' of codification and abstraction for each knowledge group. Once a particular knowledge item is scanned by an agent from the knowledge ocean, this property determines the level of codification and abstraction of the knowledge store that holds the newly obtained item. Therefore, there are knowledge groups with knowledge items that will start more codified and abstract and knowledge groups with knowledge items that will start less codified and abstract.

Structuring effort. Structuring effort is a dynamic property. It is the absolute distance that a knowledge store and its predecessors have moved through the *I-Space* in the codification and abstraction dimensions. It is an emergent outcome of the simulation, and no starting level can be assigned.

3.4 Knowledge Networks

Linked Knowledge

When configuring the simulation, the user can specify links between individual knowledge items in the knowledge ocean (also across knowledge groups). This results in knowledge networks. Agents cannot link knowledge themselves during the simulation runs, since knowledge links are defined as being natural, and discovered, rather than man-made. Links are set by the user a priori and created during the start-of-simulation processing. This allows the user to specify exactly the density of knowledge links.

Linked knowledge is important when it comes to obtaining knowledge from the knowledge ocean (cp. scanning action). The links affect the probability of scanning knowledge from the knowledge ocean in the following way: the probability of scanning a given knowledge item in the ocean is directly proportional to the number of knowledge items in an agent's possession with links to that given item. Furthermore, the codification and abstraction of the linked knowledge stores the agent possesses is directly proportional to the probability of scanning the given knowledge item in the ocean. As a result, knowledge items in the ocean that have links to items in highly codified and abstract knowledge stores are more likely to be scanned than unlinked knowledge items or items with low-codification and low-abstraction links.

From an agent's perspective, the more codified and abstract a knowledge item gets, the higher the probability that the agent who possesses that knowledge item (who holds it in his knowledge stores) obtains the knowledge items that are linked to that particular knowledge item. Ontological connections are created by the user at the beginning of the simulation. The agent tries to discover the connections (linked knowledge items) epistemologically by investing in increased codification and abstraction.

Discovered Through Investment (DTI) Knowledge

DTI knowledge is a special kind of knowledge, which is discovered through investing in related (child) knowledge. DTI knowledge items cannot be discovered through scanning the knowledge ocean. The user chooses a set of knowledge items to be children of a DTI knowledge item (DTI network; separate from linked knowledge). The only way for an agent to discover DTI items is to successfully scan the children and then to codify and abstract (or absorb and impact) them above (or below) a certain user-set value. Once the specified codification and abstraction levels are reached, the agent automatically obtains the DTI knowledge item. Investing in the child items, i.e. scanning, codifying and abstracting them, is the primary means of getting DTI knowledge. Once an agent has discovered a DTI item, it is treated like a regular knowledge item, i.e. other agents are then able to scan it from the agent that possesses it. By specifying the value characteristics of the DTI knowledge item, the user can indirectly determine the value of the respective DTI network.

3.5 Owning vs. Possessing Knowledge

An agent can own or possess knowledge. Agents possess the knowledge if they possess a knowledge store that contains the knowledge item. This happens after they have scanned a knowledge item or acquired it through licensing (see further down for details on individual action types). It can also be the result of an original endowment of knowledge the user specifies when determining the simulation settings. Possession ends when the agent finishes possessing the knowledge store that contains the knowledge item. This happens after the agent disposes of all knowledge stores containing the knowledge item, or when the agent dies (i.e. is cropped).

An agent may possess a piece of knowledge, but not own it. Ownership indicates when an agent has legal rights to benefit from a knowledge item. Ownership happens through patenting or copyrighting (of a knowledge item). For knowledge to be patentable or copyrightable, at least one knowledge store containing the knowledge item must be above a certain user-set codification and abstraction level. This means that knowledge can only be owned once a knowledge store has reached a certain level of codification and abstraction; however, all knowledge can be possessed. An agent cannot patent or copyright a knowledge item that is in the public domain. Knowledge is in the public domain if a) a certain user-specified, minimum number of agents possess it, b) it had previously been patented or copyrighted but the patent or copyright expired, or c) the user declared it to be in the public domain from the start on. Being in the public domain is a final state for knowledge items (with regards to a), this means a knowledge item will permanently be in the public domain, i.e. the knowledge item is no longer available for copyright or patent protection even if some agents dispose of it later). All of the agent's knowledge stores that hold a patented or copyrighted knowledge item are considered to be owned by that agent.

Ownership also happens when the knowledge is bought in a trade. This means, trading brings about transfer of ownership: ownership for one agent (the seller) ends and ownership for another agent (the buyer) begins. Ownership ends when the agent dies (i.e. is cropped) or when the agent disposes of all knowledge stores containing the knowledge item. It also ends when knowledge is sold in a trade or when the duration of the patent or copyright ends. Note that in these cases, though the agent loses ownership, it still possesses the knowledge.

Not all knowledge is owned, e.g. with tacit knowledge, we are more concerned with possessing than with owning. An agent possessing knowledge will be able to exploit it and learn from it whether or not that agent owns it (see further down for details). However, strengthening an ownership claim on particular knowledge, i.e. patenting or copyrighting, decreases the ability of others who only possess the knowledge to exploit it.

3.6 The Value of Knowledge

What is knowledge worth? Agents can capitalize on their knowledge by exploiting, trading or (exclusively and non-exclusively) licensing it. The base value of a knowledge item will then be transformed into a *trade amount* or *license amount* or *exploit amount*. This is done according to the level of abstraction and codification of the knowledge store, which holds the knowledge item, and according to the diffusion of the knowledge item. The user can define an industry specific table of revenue multipliers based on abstraction and codification levels. When determining amounts, the abstraction and codification level of the knowledge store is read, and then the corresponding abstraction-codification-multiplier from the industry-specific revenue-multiplier-table is looked up. In the *I-Space*, the value of knowledge is a function of both utility (how codified and abstract) and scarcity (how far diffused). Usually, the higher the codification and abstraction level, the higher the revenue multiplier, i.e. more codified and abstract knowledge is worth more. More codified and abstract knowledge, however, is also more likely to be diffused. This erodes the value of knowledge. In addition to the structure (codification and abstraction) and the diffusion of knowledge, the calculations include obsolescence. Obsolescence also has a negative effect on value: completely obsolete knowledge is worthless. Whereas the revenue multipliers depend on the characteristics of the knowledge store (level of codification and abstraction), obsolescence solely depends on the properties of the knowledge item.

Note the distinction between capital value and rental value of knowledge. When knowledge is traded, it is traded for its capital value, and this is a once off transfer of funds. In contrast, when knowledge is licensed, the revenue/cost of the license is a recurring revenue/cost, charged by the licensor, to the licensee, at the end of every period, for the duration of the license.

The *trade amount* is calculated as follows:

$$\begin{aligned}
&\text{Trade amount}\\
&= (\text{base value}) \times\ (\text{simulation-wide trade revenue multiplier})\\
&\times\ (\text{price multiplier attribute for the seller agent}) \qquad (1)\\
&\times(\text{relevant abstraction-codification-multiplier for knowledge store})\\
&\times\ (1\ \text{-}\ \text{obsolescence for knowledge item})\ /\ (\text{diffusion for knowledge item})
\end{aligned}$$

The simulation-wide trade *revenue multiplier* allows the user set the multiple to increase (or decrease) the *base value* in order to arrive at the amount for which knowledge should be traded. The *price multiplier* is used to account for the fact that certain agents routinely price knowledge higher than others do. The formula for calculating the trade amount is the basis for calculating both, the exclusive and non-exclusive license amount. The exclusive license amount is the trade amount divided by the license length. The non-exclusive license amount is the trade amount divided by the license length and divided by the number of agents that the knowledge could potentially be licensed to, i.e. the number of agents alive. Both license amounts also feature a revenue multiplier each to allow the researcher to incorporate context specific settings.

The *exclusive license amount* is calculated as follows:

$$\begin{aligned}&\text{Exclusive license amount}\\&= (\text{base value}) \times (\text{simulation-wide trade revenue multiplier})\\&\times\ (\text{price multiplier}) \times (\text{simulation-wide exclusive license revenue multiplier})\\&\times(\text{relevant abstraction-codification-multiplier for knowledge store})\\&\times(1 \text{ - obsolescence for knowledge item})\ /\\&[(\text{diffusion for knowledge item}) \times (\text{license length})]\end{aligned} \tag{2}$$

The *non-exclusive license amount* is calculated in the following way:

$$\begin{aligned}&\text{Non-exclusive license amount}\\&= (\text{base value}) \times (\text{simulation-wide trade revenue multiplier})\\&\times(\text{price multiplier})\times(\text{simulation-wide non-exclusive license revenue multiplier})\\&\times\ (\text{relevant abstraction-codification-multiplier for knowledge store})\\&\times(1 \text{ - obsolescence for knowledge item})\ /\\&[(\text{diffusion for knowledge item}) \times (\text{number of agents alive}) \times (\text{license length})]\end{aligned} \tag{3}$$

Finally, the *exploit amount* is expressed as follows:

$$\begin{aligned}&\text{Exploit amount}\\&= (\text{base value}) \times (\text{simulation-wide exploit revenue multiplier})\\&\times(\text{relevant abstraction-codification-multiplier for knowledge store})\\&\times(1 \text{ - obsolescence for knowledge item})\ /\ (\text{diffusion for knowledge item})\end{aligned} \tag{4}$$

4 Agents

The *SimWorld* is populated by virtual agents. Depending on the agent group, those agents have certain characteristics and perform numerous actions.

4.1 Agent Properties

Each agent belongs to a particular agent group. As already mentioned, the properties of those agent groups can be set by assigning distributions. The value of a particular property for a specific individual agent at the start of the simulation depends on the distributions set on the group level. Certain property values of agents will change over time as the simulation proceeds (dynamic properties). There are various properties that agents can have. They can be either *general properties* or *action specific properties* as described in the next subsections.

Note that there is no single property to allow the user to specify the density of agent knowledge. Consider that it would have been possible to have an 'agent knowledge density' property with a scale of 0 to 1, where 0 indicates that all agents know no knowledge, 1 that all agents know all knowledge, and values in between that some agents know some knowledge. This property would then have been specified for every agent group and knowledge group. Instead, the user can directly endow agent groups with a prior stock of knowledge, i.e. they can specify which particular knowledge items agents possess from period one on. In addition to that, agent-knowledge-densities come out as emergent properties of a simulation. There are various action-types that allow agents to acquire knowledge (e.g. scan, trade, license, etc.) and each agent has some propensity to indulge in that action-type. Agent-knowledge-density, then, is really synonymous with diffusion.

General Properties for Agents

Location. Agents have a specific two-dimensional location (X/Y) in the *SimWorld*. Both X and Y can have values of between 0 and 100. For simplicity, it is assumed that more than one agent and knowledge store can occupy the same coordinate in space. This rule pertains to actions such as agent relocating. If an agent re-enters the *SimWorld*, having exited earlier, its coordinate is randomly assigned.

Financial funds. An important dynamic property of agents are their *financial funds*. The financial funds of each agent are recomputed at the end of each period. The 'per period carrying cost' for all knowledge items held by the agent *during* a period is subtracted from the agent's financial funds. The exact amount subtracted depends on the abstraction and codification of that knowledge. As seen before, knowledge with a higher level of codification and abstraction has a lower carrying cost (higher carrying gain).

Note that financial funds are also updated during each action: costs of action are subtracted, and the revenues from the action are added and payments resulting from the action are subtracted. The licensing costs and revenues (license amount for each knowledge item) are also subtracted from (licensee) or added to (licensor) an agent's financial funds at the end of each period for the length of the license. Remember that license amounts are per period, whereas trade amounts are once off. At the end of each period, the agent's income per period (i.e. income from external sources, not relating to specific transactions) is added to its financial funds, and the agent's expenditure per period (i.e. expenses from external sources, not relating to specific transactions) is subtracted from its financial funds. Agents currently cannot go into credit, i.e. agents cannot have negative financial funds. Agents are cropped (die) if they run out of funds.

Experience funds. The experience funds are not changed as part of a transaction (like financial funds for example after a trade), but whenever an action is undertaken. After every action, the base experience gained is added to the experience funds of an agent. Agent experience has the following effect: an agent can only perform an action when the user-set experience threshold for that action has been passed, i.e. the agent must have accumulated enough general experience to undertake an action. Experience funds are not updated at the end of each period; they are only changed during/after each successful action.

Income per period. It is the base income per period from external sources, not relating to specific transactions, i.e. it does not include money made from trading, licensing, and other transactions. Thereby, income per period is merely a fixed, recurring income.

Expenditure per period. Agents can have base expenses per period. Those do not relate to specific transactions, i.e. they do not include costs of taking actions, or costs relating to trading, licensing, and other transactions. Thus, expenditure per period is merely a fixed, recurring cost.

Relocate distance. The relocate distance determines how far an agent moves when it performs the relocate action.

Vision. Vision determines how far the agent can see spatially. An agent's vision is a certain radius from its current location within which it can scan and call for *meetings* (see further down for detailed description of those actions). Thereby, it establishes the size of the market within which the agent operates. Some will be village markets and some will be global markets.

Price multiplier. Some agents habitually sell at a higher price than other agents do. Hoarders have a high price multiplier - greater than one - whereas Sharers have a low price multiplier - less than one. The price multiplier characteristic is normally assigned a value between 0.5 and 1.5.

Activity rate. Activity rate is the number of actions that an agent takes per period. By setting the activity rate, users can create groups that work frantically (many actions per period) and groups that are laid back (with few actions per period). To create agents that work-hard-and-play-hard, one can set the activity rate high, but set the propensity for the relax action high as well, meaning that the agent will relax often. Whereas *vision* affects the range of an agent's interaction, one could say activity rate affects the rate of interaction.

H-Factor. Users can modify the behavior of agents in their choice of meeting partners. It creates the possibility of specifying whether agents are interested in meeting other agents similar or dissimilar to them, in terms of their knowledge portfolio. The H-Factor determines the level of homophily, i.e. the preference for meeting agents conditioned on the similarity of knowledge portfolios. It can take values between minus one and one, and it is used to influence the kinds of agents that an agent will seek when it is initiating a meeting request[1]. The sign of the H-Factor determines whether an agent is homophilic or heterophilic and the magnitude of the H-Factor determines how exclusive or discriminating an agent is when it comes to meeting others. An agent with an H-Factor of zero is neutral and will accept to meet all visible agents with equal probability.

Propensity to scan from ocean. When an agent decides to scan, it scans either from the knowledge ocean or from the knowledge stores and agents surrounding him. The propensity to scan from ocean specifies the probability of scanning from the knowledge ocean. It should be between zero and one.

Action Specific Properties for Agents - Agent Group Level

Propensity. Propensity refers to the likelihood of performing a particular action. Propensity is specified for each action-type. Every time an agent wants to perform an action, it must decide what particular action to take. The agent will attempt to embark on a given action-type with the probability drawn from the distribution associated with the propensity property for that action type for the respective agent group. That is, propensity is the probability of the agent attempting a particular action, each time the agent takes an action. Again, agents may take zero or more actions per period, but this depends on the settings for the activity rate. The propensities across all action-types for a given agent should sum to one[2]. If the sum comes out to more than one or less than one,

[1] To understand this better, consider the following example. An agent has an H-Level of 0.6. A positive H-Level indicates that the agent is homophilic, and therefore prefers to meet agents with similar knowledge portfolios. Furthermore, a value of 0.6 indicates that this agent will only consider meeting agents that are at least 60% similar. When the agent is about to initiate a meeting request, it will first examine all the agents in its visible range and assess a degree of similarity of knowledge portfolio to each. Let us say he comes across the following agents: Agent 12 - Similarity 45%; Agent 13 - Similarity 65%; Agent 14 - Similarity 85%; Agent 15 - Similarity 50%. In this case, he will immediately reject agents 12 and 15 since they are not similar enough. Now Agents 13 and 14 will be considered acceptable, and will have an equal probability of being met, which will be determined by a random draw. Let us say he chose agent 13. Now the meeting is proposed, and Agent 13 goes through a similar procedure to ensure the initiating agent is similar (or dissimilar) enough to meet based on agent 13's H-Level. Just as a positive H-Level defines a minimum level of similarity that must be met before an agent can be considered for meeting, a negative H-Level specifies the maximum level of similarity between two agents that can exist if they are to meet. In this case, the agent would be considered heterophilic. So for example, an H-Level of - 0.75 says that this agent will only meet other agents who are at least 75% dissimilar to it (or at most 25% similar) in terms of their knowledge portfolio.

[2] For example, assume three actions: Buy (propensity to buy = 0.6), Sell (propensity to sell = 0.2), and Codify (propensity to codify = 0.5). The total is 1.3, which has to be adjusted. All propensities for this agent need to be divided by 1.3, so that the propensities sum to 1. This is the reason for why, when it comes to creating distributions, each propensity attribute should have a constant distribution and the sum of the constant distribution values for all the propensity properties for a particular agent group should sum to one.

then the program will scale the propensities to one. Another way of looking at the propensity setting is that for all the different actions an agent is allowed to perform during a simulation, the propensities determine the proportions each action-type will be carried out. Propensities do not determine whether an agent will execute an action successfully or not. This is influenced by the next property, *effectiveness.*

Effectiveness. Effectiveness is used to determine the likelihood of succeeding in a particular action-type (value between 0 and 1, 1 being most effective). Whether an action succeeds depends on the effectiveness of all agents participating in the action. The average of the effectiveness of all agents participating in the event gives the probability of the action succeeding, given the action is undertaken. It is assumed that the environment does not introduce any additional random variation.

Efficiency. Efficiency is an agent-group specific weighting factor used to adjust the cost of performing a particular type of action. In other words, it is a cost modifier for an action. The more efficient (value between 0 and 1, 1 being most efficient), the lower the costs that are incurred when conducting an action.

Action Specific Properties for Agents - Simulation Level

Whereas the aforementioned agent properties are set on the agent group level, there are also properties that the user specifies on the simulation level. Consequently, they are the same for all agents regardless of what agent group they belong to.

Patent/Copyright length and strength. The user can set the number of periods a patent lasts and specify the strength of it. The same applies for copyright. The strength of a patent or copyright has an influence on whether agents who possess but not own a particular knowledge item can exploit it. The value for strength should be between zero and one. It influences an agent's effectiveness of exploiting knowledge that has been patented or copyrighted. Agents that do not have the patent or copyright for a particular knowledge item or have not licensed it should be less likely to succeed in exploiting the knowledge.

Exclusive/Non-Exclusive license length. Exclusive and non-exclusive licenses both have a particular user-set length, measured in number of periods.

Similarity function. The similarity function determines the method for calculating the similarity between two agents in terms of their knowledge portfolios. The simulation allows the notion of similarity to be specified in two unique ways: *exclusive (0)* and *inclusive (1).* Under the *Exclusive* method, similarity is only considered to be the total of those knowledge items that both the agents possess divided by the sum of the unique items that they possess considered together[3]. Under the *Inclusive* method, similarity not only includes the

[3] So if Agent 1 has knowledge items 2, 4, 5 and 6, while Agent 2 has knowledge items 1,2 and 3, the similarity in this case is calculated as the total of the items they both possess (only item 2), i.e. 1, divided by the total number of unique items possessed by them together (items 1,2,3,4,5 and 6), i.e. 6. Hence, the similarity is 1/6 or 16.66%.

knowledge items they both possess, but also the knowledge items they both do not possess[4].

Copyright/Patent codification and abstraction threshold. An agent can only patent knowledge that it holds in a knowledge store above a certain user-set codification and abstraction threshold. This also applies to copyrights.

Public domain absolute diffusion threshold. Once a certain number of agents possess a knowledge item (knowledge is diffused to a set of agents), it is considered to be in the public domain and cannot be copyrighted or patented anymore.

Scan from surroundings function. This simulation level property, which will take values of either 1 or 2, helps determine the particular set of knowledge stores an agent scans from within its vision. The two possible methods of selecting a store to scan are described below (see scanning action).

Trade, exploit, exclusive license, non-exclusive license revenue multipliers. These four parameters let the user specify at which multiple of the base value the agents try to capitalize on their knowledge through the trading, exploiting, and (exclusive and non-exclusive) licensing actions.

Base financial cost. It is the base cost of participating in a particular action. Every action has a financial cost, which must be paid for by the agent when embarking on that action. The agent must pay this cost, from its financial funds, immaterial of whether the action succeeds or not. Financial funds are updated during each action, i.e. costs of action are subtracted. If agents have insufficient funds to undertake an action, then the action cannot be successfully taken, and their financial funds are not depleted. The final cost particular to an agent when undertaking an action equals the base financial cost for that action multiplied by one minus the agent specific efficiency for that action.

Base experience gained or lost. It is the amount of experience gained, or lost, by participating in an action-type. Each action undertaken by the agent normally increases the agent's base experience by a certain amount. There is only one experience fund per agent, no accumulated experience per action-type.

Experience threshold required. It is the minimum amount of experience required to undertake a specific action.

4.2 Actions

The agents in the simulation are able to perform various *actions* and thereby to adopt different role types. All actions - aside from entering - require the agent to be inside the *SimWorld*. Actions are assumed to have zero duration, start and end in the same period and are purposefully taken by agents. This stands in contrast to states that have some duration and that are inferred by the system

[4] In this case, carrying forward the previous example, if there was a total of 10 knowledge items in the simulation, then the similarity between agent 1 and 2 would be the $(a+b)/c$ where a is the number of items they both have, b is the number of items they both don't have and c is the total number of items in the simulation. This works out to be, $a = 1$ (they both have item 2), $b = 4$ (they both do not have items 7 through 10) and $c = 10$. Therefore the similarity is $(1+4)/10 = 50\%$.

based on the actions taken in the previous periods. For example, the system will infer that the agent starts *owning* a knowledge item if the agent bought the knowledge item during the period, similarly the system will infer that the previous owner finishes owning the knowledge upon selling ('owning' is a state).

An agent will be permitted to undertake an action, if and only if, the environment and the agent state permit the action. Most importantly, an agent can only perform an action, if it has sufficient financial funds and experience. An agent will not be able to undertake an action if it cannot pay the cost of that action. Entry into the *SimWorld* is only possible if the agent is outside the *SimWorld* and alive. In addition, agents can only license or trade a piece of knowledge if they own the knowledge item.

The state of the world as well as that of the agent (and the knowledge) changes after an action is successfully undertaken. What follows next is a description of each action-type. When deciding what to do in a period, agents pick from this list of actions.

Scanning (storing). An agent can scan knowledge. Scanning means picking up a random knowledge item, whether from the knowledge ocean or from other agents' knowledge stores. The probability of scanning from the knowledge ocean is specified by the agent-group-level property 'Propensity To Scan From Ocean'. An agent can scan any knowledge item in the knowledge ocean, but can only scan knowledge items in knowledge stores within its vision. An agent can scan knowledge possessed or owned (patented or copyrighted) by other agents within its vision. Agents only try to pick up knowledge items that they do not already possess at that level of abstraction and codification. If a knowledge item is successfully scanned, it starts off in a new knowledge store possessed (but not owned) by the agent. Depending on the origin of the knowledge item, the new knowledge store picks up the level of codification and abstraction from the knowledge group the knowledge item belongs to (knowledge item from knowledge ocean) or from the knowledge store it found the item in (knowledge item from other agent). If the agent fails to find a store he does not already know (has a store with the same codification level and abstraction level) then the action will fail and the agent will lose his turn.

Remember the special role of networks. The more an agent codifies and abstracts the knowledge stores of a particular knowledge item, which is linked in the knowledge ocean, the more likely it is that the agent will obtain, through scanning in the ocean, those linked knowledge items. Also remember that once an agent has scanned all the child items of a DTI knowledge item and has codified and abstracted them up to a certain level, then the agent automatically gets the DTI knowledge item associated with that DTI network. The action that is triggered if all the conditions for obtaining a particular DTI are met (set of knowledge items, codification and abstraction threshold) is called *discover*.

If the agent ends up scanning from the surrounding knowledge stores, there are two possible methods for selecting the store to be scanned. The method is specified by the simulation level property 'Scan From Surroundings Function'.

i. All Stores and Agents (1): All knowledge stores in the agent's vision and all knowledge stores possessed by agents in this agent's vision are considered (but only the knowledge stores that the agent does not already possess). The ease with which knowledge is scanned is a function of the degree of codification and abstraction of the knowledge involved. Knowledge items in knowledge stores with higher codification and abstraction have a higher propensity of being scanned, so the program weights the probability of being scanned by the codification and abstraction level of the knowledge chosen.
ii. Agents Only (2): The agent only considers the knowledge stores of visible agents that he would want to meet (according to its H-Factor). He pools together all the knowledge stores of the agents in his vision who would meet his H-Factor requirement (only the knowledge stores that the agent does not already possess); from this pool, he then selects a knowledge store. The probability of scanning a knowledge store is again the codification and abstraction of this knowledge store divided by the total codification and abstraction of the knowledge stores in the pool. If no suitable stores are found, then the agent will fail in his attempt to scan.

Codifying. An agent can codify knowledge. Codification only occurs on knowledge stores (form), not on knowledge items (content). The agent must possess the knowledge store to carry out codification. Each codification action creates a new string-of-pearl store with an increased level of codification. Codification of the new knowledge store increases by the codification increment specified for the knowledge item in the store. The level of codification cannot exceed one.

Abstracting. An agent can abstract knowledge. Again, abstraction only occurs on knowledge stores, not on knowledge items. The agent must possess the knowledge store to carry out abstraction. Each abstracting action creates a new string-of-pearl store with an increased level of abstraction. Abstraction of the new knowledge store increases by the abstraction increment specified for the knowledge item in the store. The level of abstraction cannot exceed one.

Absorbing and impacting. We speak of absorption if an agent's knowledge gets more uncodified, and of impacting if an agent's knowledge gets more concrete. They can be considered as negative codifying and abstracting actions. However, the user can set separate characteristics for each of them.

Patenting. An agent can patent knowledge for a certain duration and with a specific strength. An agent can only patent a knowledge item it possesses, and only if it holds the knowledge item in a knowledge store that has an abstraction and a codification level above a user-set level. That is, an agent can usually only invest in patenting after it has invested in codifying and abstracting. Each patent has a particular strength and duration. The user can assign a distribution for both characteristics (general setting for all knowledge). Once an agent patents an item, it owns that item. Consequently, all of the agent's knowledge stores that hold the newly patented knowledge item are then eligible for the actions that require ownership (trading, licensing, etc.). An agent may not patent a knowledge item that is already possessed by a user-defined number of other agents (diffusion threshold). This is because knowledge that is in the public domain cannot be patented or copyrighted. ‘In the public domain’ is defined as follows. First, ’in the public domain’ means that other agents also possess the knowledge item in question. Knowledge that is widely diffused cannot be patented or copyrighted, and it is up to the user to specify what widely diffused means by setting an appropriate level of absolute diffusion. Once a patent or copyright is requested for a knowledge item that has surpassed the diffusion threshold, that knowledge item will permanently be in the public domain, i.e. the knowledge item is no longer available for copyright or patent protection. The threshold is specified as the minimum number of agents that must hold the knowledge item in order for it to be considered public domain. Second, all knowledge items with expired copyrights or patents automatically become public domain. Third, the user can opt to put all knowledge items of a group into the public domain. This means, from period one on, these knowledge items will be in the public domain and cannot be patented or copyrighted during the simulation.

Copyrighting. An agent can copyright knowledge for a certain duration and with a specific strength. An agent can only copyright a knowledge item it possesses, and only if it holds the knowledge item in a knowledge store above a certain level of codification and abstraction. Each copyright has a particular strength and duration. The user can assign a distribution for both characteristics (general setting for all knowledge). Once an agent copyrights an item, it owns that item. Consequently, all of the agent’s knowledge stores that hold the newly copyrighted knowledge item are then eligible for the actions that require ownership (trading, licensing, etc.). An agent may not copyright a knowledge item that is in the public domain (see patenting). Patents or copyrights are not diffusion blocking. They just restrain other agents from doing certain things with their knowledge items, e.g. selling, licensing, successfully exploiting.

Learning. An agent can learn, i.e. register, existing knowledge. Learning enables agents to exploit the knowledge items they learned of. This means that before knowledge items can be exploited, learning has to take place. Agents can only learn from a knowledge store they possess. The more string-of-pearl knowledge stores an agent possesses for a particular knowledge item, the more probable it is that this knowledge item will be learned first. An agent’s chance of successfully learning is higher for more codified knowledge.

Exploiting. An agent can exploit knowledge to gain value. Exploitation means capitalizing on internalized knowledge. This means that an agent must register the knowledge prior to exploiting it, i.e. perform the learning action on the knowledge item. The financial funds of the exploiter agent are increased by the value of the exploiting. Exploiting increases the financial funds of the agent by the intrinsic *base value* of the knowledge item multiplied by the *exploit revenue multiplier*. The level of codification and abstraction, the degree of diffusion, and obsolescence are also taken into account as described earlier. An agent can try to exploit a piece of knowledge that it possesses, even if it does not own it. The higher the strength of the patents and copyrights held on the knowledge by other agents, the lower the probability of successfully exploiting, if the agent does not have a license for that knowledge. If the agent holds the patent or copyright or has a license for the knowledge, then their chance of successfully exploiting the knowledge is high (patent strength considered to be zero for those agents).

Meeting. An agent can meet with another agent. Only agents who have initiated the meeting (initiator) and those who have responded positively (responder) are allowed to attend. An agent can only initiate a meeting with agents within its vision. Meeting is a prerequisite for a trade or a license. Based on the H-Factor setting, agents chose meeting partners at random or just select meeting partners that are similar or dissimilar to them. In the simulation setup, the user can specify a network of agents that have already met (prior acquaintances). Once two agents have met, they will not try to initiate another meeting with each other for the rest of the simulation run.

Buying knowledge and selling knowledge (trading). An agent can buy (sell) knowledge from (to) another agent for a certain price (sale amount). In contrast to scanning, buying only targets knowledge that is owned by other agents. Meeting is a prerequisite for trading, and mutual consensus is necessary. Agents can only sell knowledge stores that they own, i.e. knowledge stores with a knowledge item that is copyrighted or patented. This also implies that if one wants to trade knowledge, it must be up to a certain level of codification and abstraction (depending on the patent and copyright threshold). The buyer acquires ownership and the seller loses ownership. This means that the patent or copyright for the underlying knowledge item is terminated for the seller, and the rest of the patent or copyright (remaining time) is transferred to the buyer. Note that the seller still possesses the knowledge and is still in a position to learn and to exploit it. Only knowledge that the acquiring agent has not previously owned will be traded. A knowledge store is only transferred to the buying agent if its level of codification and abstraction is different to the level of other knowledge stores in its possession that hold the same item. However, the patent or copyright for the underlying knowledge item is always transferred. The trading price is determined using the formula for trade amount described earlier. The financial funds of the seller agent are increased by the sale value for the trade, and the financial funds of the buyer agent are decreased by the sale value for the trade.

Exclusive licensing and non-exclusive licensing. An agent can license knowledge to other agents for a certain per period license amount/fee. Meeting is a prerequisite for licensing, and mutual consensus is necessary. Agents can only be a licensor for knowledge that they own. The licensor grants a license to a particular knowledge item to a licensee. In the case of exclusive licensing, only one (exclusive) license can be given. In contrast, the non-exclusive licensor gives the license to a knowledge item to various licensees for joint possession. That is, multiple non-exclusive licenses may be given.

The licensee does not acquire ownership of the knowledge. The license is transferred as well as a random knowledge store holding the knowledge item (but only if the licensee does not already possess the store with that level of codification and abstraction). Though they own the knowledge, exclusive licensors will have difficulties exploiting the knowledge - only the licensees are entitled to exploit the knowledge. When it comes to exploiting, exclusive licensors are treated like all the other agents that do not have a license: the patent or copyright strength will determine the success of their exploit action; the special type of license where exclusive licensors can continue to exploit their knowledge is omitted from the simulation. However, the non-exclusive licensor is treated like if he has a non-exclusive license, i.e. patent strength will be zero for him as well.

At the end of each period, and for the duration of the license, the financial funds of the licensor agent are increased by the license amount. Similarly, for the duration of the license, and at the end of each period, the financial funds of the licensee agent are decreased by the license amount. The price of the exclusive and of the non-exclusive license, i.e. the license amounts, are each determined using the formula for license amount shown earlier.

Disposing. Agents can dispose their knowledge stores - one store with underlying item per dispose action. Single knowledge items cannot be disposed since these are eternal propositions.

Relocating. An agent can relocate a certain distance. Relocating implies moving closer to/further from other agents or knowledge stores. The distance an agent moves per relocation depends on the distance setting for the relocate action (agent group property); the direction each time the agent relocates is random. Relocation is important in the context of scanning and for calling a meeting as it affects what one sees and whom one sees. As agents can only scan and call for meetings within the radius of their vision, they are only able to pick up knowledge or meet people in a different area by moving/relocating. When agents relocate, they leave their knowledge stores behind, but can still use them (N.B.: A new knowledge store is always given the same location as the agent.).

Exiting. An agent can exit the *SimWorld.* It can do so in any particular period with its current rent earnings. An example of exiting is retiring to the Bahamas to live a life of luxury. Once the agent has exited, the only action possible for the agent is entering.

Entering. An agent can enter the *SimWorld.* Obviously, the agent must be outside the simulation world to undertake this action. Being in *SimWorld* is a particular state an agent has or has not. It indicates when an agent is alive and is a citizen of the *SimWorld* (i.e. has not retired).

Relaxing. An agent can relax for a specific duration. Relaxing means that the agent will undertake no actions. The duration specifies the number of periods the agent is relaxing for (e.g., if the agent is relaxing with duration 2.4, then the agent is doing nothing for two periods of the simulation). Actions are not queued while agents are relaxing - they simply do not try to undertake any actions whatsoever. Agents will only start trying to undertake actions after their relaxation duration is over. Relaxing is currently the only action-type that has a duration. The simulation determines the duration randomly by assigning a value between 1 and 10.

The death of an agent - cropping. Agents die if there are insufficient resources to sustain them. The agent's financial funds must be zero or negative for the crop action type to occur. Cropping happens each period after the updating of financial funds, which happens during end-of-period processing. Once the agent is cropped, no action is possible for the agent, since the agent is dead. After cropping, states like 'owning' and 'is in *SimWorld*' are updated, because a dead agent can no longer own anything or be in the *SimWorld*. All agents with financial funds of zero must be cropped. This goes almost without saying since an agent without financial funds cannot afford to perform any action, as they cannot afford the costs of performing any action. However, the simulation still needs to crop/kill them, so the user can see which agents died and in what period. It is assumed that dead agents cannot have an estate, and that they merely forfeit ownership when they die, rather than have ownership vest in the estate of the deceased agent. Being dead or alive is a particular state an agent has.

5 Validation

Can *SimISpace2* capture the basic tenets that *I-Space* theory puts forward? The *I-Space* [1,2] is a conceptual framework that relates the degree of structure of knowledge (i.e. its level of codification and abstraction) to its value and diffusibility as that knowledge develops. To check whether the rules and architecture of the *SimISpace2* environment fully implement this relationship, we can design and run a couple of test cases, the results of which will have to match the expected *I-Space* behavior the theory predicts.

5.1 More Structured Knowledge Diffuses Faster

Tacit knowledge (low codification and abstraction) flows very slowly between agents and often only in face-to-face situations. Codified and abstract knowledge, by contrast, can diffuse rapidly throughout a population, whether such diffusion is desired or not [1,2]. To test for this, we create three knowledge groups of equal size, one with a low level of codification ('KG1' – 0.1), one with a medium level of codification ('KG2' – 0.5), and one with a high level of codification ('KG3' – 0.9). What we expect to see is that the agents that scan knowledge from other agents pick up the more codified groups faster than the less codified groups. This is confirmed in the simulation results displayed in Fig. 4. 'KG3', the most codified group, is scanned fastest, followed by 'KG2'. The least codified group, 'KG1', is picked up last and thereby lags behind[5].

5.2 Extracting Value from Knowledge

The development of knowledge from tacit to codified and abstract can add value to knowledge to the extent that it does not diffuse to other agents, and to the extent that it succeeds in shedding noisy data without simultaneously shedding usable information. Thus, extracting value from knowledge is difficult. The

[5] To make the test cases as uncomplicated as possible, we only vary the level of codification. However, comparable results (not reported in this paper) were obtained when experimenting with different levels of abstraction.

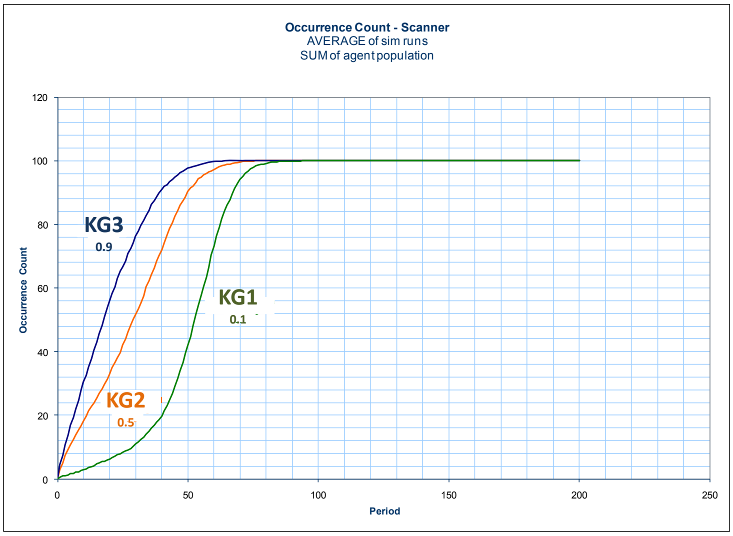


Fig. 4. More structured knowledge diffuses faster

structuring of knowledge increases its *utility* while - on account of its increased diffusibility - simultaneously compromises its *scarcity* [1,2].

More Diffused Knowledge Is Less Valuable

The more diffused knowledge is the less valuable it should be to agents that possess it. To test for this, we create three agent groups of equal size and two knowledge groups of equal size. The two knowledge groups have exactly the same properties; one is exclusively assigned to the agent group we call 'AG1', and one is assigned to the other two agent groups, 'AG2' and 'AG3'. AG1 and AG2 both exploit their knowledge and do this with exactly the same frequency. The only difference is that the knowledge AG2 is exploiting is more diffused (the 'AG3' group also possesses this knowledge) than the knowledge AG1 exploits. The expected outcome is that AG1 earns more than AG2, because the knowledge it is capitalizing on is scarcer. This is exactly what our simulation results in Fig. 5 show. The accumulated financial funds in period 100 of AG1 are around 49,000, whereas the financial funds of AG2 are only around 25,000.

More Structured Knowledge Is More Valuable

More structured knowledge diffuses more rapidly and the more diffused it gets the less value can be extracted from it. However, holding everything else constant, more structured knowledge should also be more valuable than unstructured knowledge as its utility is higher. To test for this, we take the previous scenario, leave all the settings the same, but change the codification level of the two knowledge groups. Whereas in the previous scenario the knowledge had a static codification level of 0.3, we now set the codification level to 1. In this new test case, AG1 and AG2

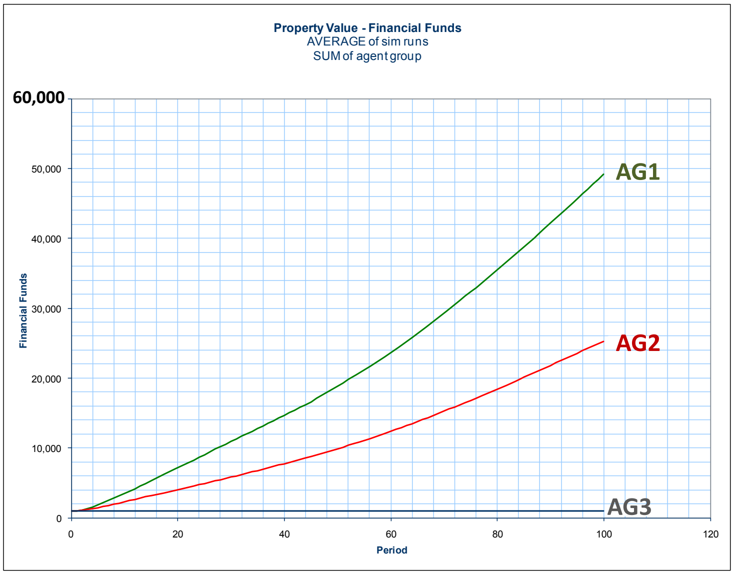


Fig. 5. More diffused knowledge is less valuable

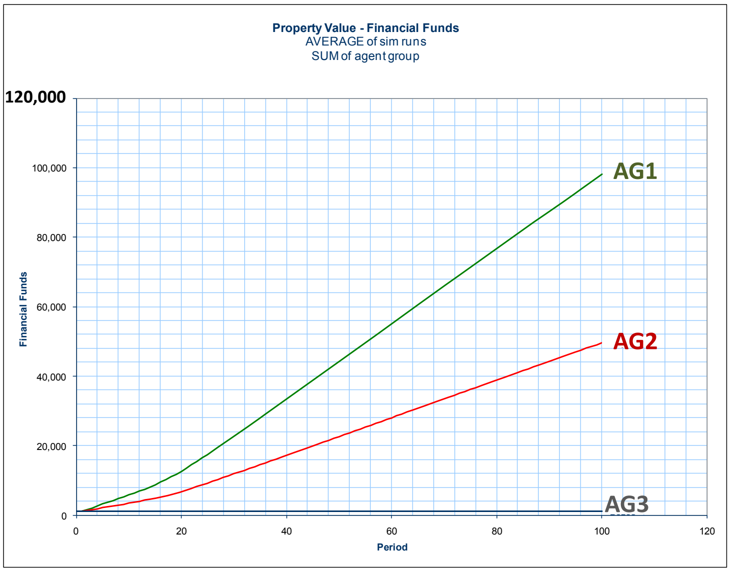


Fig. 6. More structured knowledge is more valuable

should earn more, because the utility of the knowledge they are capitalizing on increased. This is what our simulation results in Fig. 6 show. Exploiting the more codified knowledge, the financial funds of AG1 now reach 98,0000 and the financial funds of AG2 50,000.

5.3 Validation Results

The three simple test cases described above confirm that the basic theoretical relationships of the *I-Space* are properly implemented in the *SimISpace2* environment. The special interactions between the structure, the diffusion, and the value of knowledge are taken into account to enable the researcher to adequately model knowledge assets and explore agents' strategies in a knowledge economy. Any simulation model that will be implemented with *SimISpace2* will have the dynamics of a knowledge economy built in, and agent behavior can be analyzed accordingly. These dynamic effects are particularly interesting to study when working with large and complex simulations, composed of many agents and knowledge assets, as done with recent applications of *SimISpace2* [10,11].

6 Conclusion

Having described the special features of this unique agent-based graphical simulation environment in detail, the informed reader can see that *SimISpace2* is a powerful tool with which to simulate strategic knowledge management processes. The user can model and analyze very complex scenarios to study agents' actions and interactions, together with knowledge assets and their life cycles. There are numerous practical, real-world issues that can be simulated using *SimISpace2*. Examples of practical applications would be intellectual property rights policies, cognitive arms races (e.g. two pharmaceutical firms racing to discover a new molecule), countries competing to attract intellectual capital, mergers and acquisitions, intra-firm technology transfer strategies, and science, technology and innovation policies. In recent work, the tool has been used to build an application-specific model for the field of entrepreneurship. It models entrepreneurial knowledge appropriation and development and enables researchers to simulate different opportunity recognition strategies and to analyze their micro and macro effects [10,11]. Projects are under way to apply *SimISpace2* in other areas of management research, in particular studying open innovation strategies and absorptive capacity processes.

Acknowledgements. This research project was supported by the Snider Entrepreneurial Research Center at the Wharton School of the University of Pennsylvania and by I-Space Institute, LLC. The author would like to thank Ian MacMillan and Max Boisot for their help in developing the conceptual underpinnings and Ryan Namdar and Danish Munir for their excellent programming work on *SimISpace2*. An earlier version of this paper was presented at the European Conference on Modelling and Simulation (ECMS 2007, Prague, Czech Republic).

References

1. Boisot, M.H.: Knowledge assets - securing competitive advantage in the information economy. Oxford University Press, New York (1998)
2. Boisot, M.H.: Information space: a framework for learning in organizations, institutions and culture. Routledge, London (1995)
3. Boisot, M., Canals, A., MacMillan, I.C.: Neoclassical versus Schumpeterian approaches to learning: a knowledge-based simulation approach. In: Müller, J.-P., Seidel, M.-M. (eds.) Proc. of the 4th Workshop on Agent-Based Simulation (Society for Modeling and Simulation International). SCS-European Publishing House, Erlangen (Germany)– Montpellier, France (2003)
4. Boisot, M., MacMillan, I.C., Han, K.S.: Property rights and information flows: a simulation approach. Journal of Evolutionary Economics 17(1), 63–93 (2007)
5. Canals, A., Boisot, M., MacMillan, I.: The spatial dimension of knowledge flows: a simulation approach. Cambridge Journal of Regions, Economy and Society 1(2), 175–204 (2008)
6. MacMillan, I.C., Boisot, M., Abrahams, A.S., Bharathy, G.: Simulating the knowledge transfer dilemma: lessons for security and counter-terrorism. In: Proc. of the 2005 Summer Computer Simulation Conference (SCSC 2005), Philadelphia, PA (2005)
7. Ihrig, M., Abrahams, A.S.: Breaking new ground in simulating knowledge management processes: SimISpace2. In: Zelinka, I., Oplatková, Z., Orsoni, A. (eds.) Proc. of the 21st European Conference on Modelling and Simulation (ECMS 2007), Prague (2007)
8. Gilbert, N.: Agent-based models. Sage Publications, London (2008)
9. Hillier, F.S., Hillier, M.S., Hillier, M.: Introduction to management science, 2nd edn. McGraw-Hill/Irwin, New York (2002)
10. Ihrig, M., MacMillan, I., Zu Knyphausen-Aufseß, D., Boisot, M.: Knowledge-based opportunity recognition strategies: a simulation approach. In: Proc. of the Strategic Management Conference, Rome, Italy (2010)
11. Ihrig, M.: Simulating entrepreneurial strategies in a global knowledge economy. In: Proc. of the 25th European Conference on Modelling and Simulation, ECMS 2011 (2011)

Cloud Search Engine for IaaS

Yuuta Ichikawa[1] and Minoru Uehara[2]

[1] CyberAgent, Inc., Japan
ti070043@gmail.com
[2] Toyo University, Japan
uehara@toyo.jp

Abstract. Cloud based online storage enables the storage of massive data. In these systems, a full text search engine is very important for finding documents. In this paper, we propose a distributed search engine suitable for searching a cloud. In our previous work, we developed a distributed search engine, the cooperative search engine (CSE). We now extend the CSE to search clouds. In a cloud, elasticity and reliability are important. We realize these by employing consistent hashing for distributed index files in the cloud. In this paper, we describe the improved CSE architecture and its implementation for a cloud search.

Keywords: Cloud, Cooperative Search Engine(CSE), Infrastructure as a Service(IaaS).

1 Introduction

Recently, cloud computing has become very popular. A cloud is a computing service that utilizes computing resources on demand in an elastic fashion. Well known clouds are Amazon Web Services (AWS) as an IaaS (Infrastructure as a Service), Google App Engine (GAE) as a PaaS (Platform as a Service), salesforce.com as a SaaS (Software as a Service), and so on. The features of these clouds have been well defined by the National Institute of Standards and Technology (NIST)[1]. In particular, elasticity is an important concept in a cloud. In an elastic cloud, the number of nodes changes dynamically on demand. As such, the system administrator can reduce the total cost of ownership of the system by minimizing the number of required nodes.

One of the applications of a cloud is online storage. Several cloud based online storage services, such as DropBox, SugarSync and ZumoDrive, have already been developed. A feature of these online storage systems is that they can store and manage massive amounts of data. In such systems, a full text search is required to manage documents efficiently.

We aim to develop a full text search engine for an IaaS based cloud. In this paper, we propose a distributed search engine for the purpose of searching for documents stored in an IaaS based cloud. In this system, documents are stored in a storage server in the cloud. A distributed search engine is organized as many small search engines that work as an integrated large search engine through cooperation. The overall search process allows each small search engine to read the local documents in a storage server and create an index thereof.

In our previous work, we developed a distributed search engine called the CSE (Cooperative Search Engine). In this paper, we propose a new architecture based on the CSE for an IaaS cloud. The proposed search engine is suitable for elastic scale-out, i.e., dynamic changes in the system configuration. Thus, it allows scalability in performance and reliability against node failure.

This paper is organized as follows. In Section 2, we introduce several related works, particularly the CSE, which forms the basis of this research. In Section 3, we propose a new architecture suitable for a cloud and describe its design and implementation. In Section 4, we evaluate the characteristics of the proposed system. Finally, we give our concluding remarks.

2 Related Works

2.1 Cooperative Search Engine

First, we explain the basic idea of the CSE. To minimize the update interval, every web site basically creates indices via a local indexer. However, these sites are not yet cooperative. Each site sends the information about what (i.e., which words) it knows to the manager. This information is called Forward Knowledge (FK), and is meta knowledge indicating what each site knows. FK is the same as FI in Whois++[2]. When searching, the manager informs the client which sites have documents that include words in the query, and then the client sends the query to all these sites. In this way, since the CSE needs two-pass communication for searching, the retrieval time of the CSE is longer than that of a centralized search engine.

The CSE includes the following components (see Figure 1).

- **Location Server (LS):** This manages FK exclusively. Using FK, the LS performs Query based Site Selection which is described later. The LS also has a Site selection Cache (SC), which caches results of the site selection.
- **Cache Server (CS):** This caches FK and retrieval results. The LS can be thought of as a top-level CS. It realizes the “Next 10” searches by caching retrieval results. Furthermore, it realizes a parallel search by calling the LMSE described below in parallel.
- **Local Meta Search Engine (LMSE):** This receives a query from a user, sends it to the CS (User I/F in Figure 1), and performs the local search process by calling the LSE described below (Engine I/F in Figure 1). It works as a meta search engine that abstracts the differences between the LSEs.
- **Local Search Engine (LSE):** This gathers documents locally (Gatherer in Figure 1), makes a local index (Indexer in Figure 1), and retrieves documents using the index (Engine in Figure 1). In the CSE, Namazu [3] can be used as an LSE. Furthermore we are developing an original indexer designed to realize high-level search functions such as a parallel search and phrase search.

Namazu[3] is widely used as a search service on various Japanese sites. It is developed in C and Perl. Recently, Java based search engines such as Apache Solr[12] become popular.

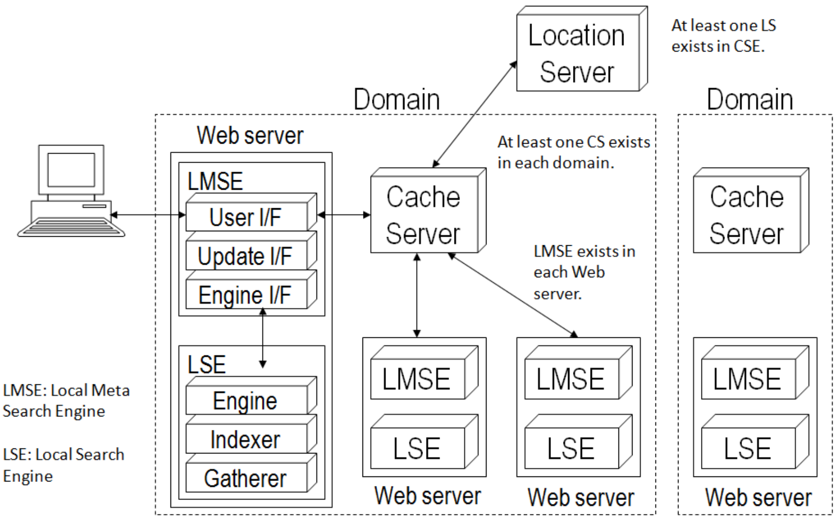


Fig. 1. The overview of Cooperative Search Engine

Next, we explain how the update process is done. In the CSE, Update I/F of the LSE carries out the update process periodically. The algorithm for the update process in the CSE is as follows.

1. The Gatherer in the LSE gathers all the documents (Web pages) in the target Web sites using direct access (i.e., via NFS) if available, or archived access (i.e., via CGI) if it is available and direct access is not, or HTTP access otherwise. Here, we explain archived access in detail. In archived access, a special CGI that provides mobile agent place functions is used. A mobile agent is sent to that place. The agent archives local files, compresses them, and sends them back to the gatherer.
2. The Indexer in the LSE creates an index for the gathered documents by parallel processing based on the Boss-Worker model.
3. Update phase 1: Each $LMSE_i$ is updated as follows.
    a. The Engine I/F of $LMSE_i$ obtains from the corresponding LSE the total number N_i of all the documents, the set K_i of all the words appearing in the documents, and the number $n_{k,i}$ of all the documents that include word k, and sends these to the CS together with its own URL.

   b. The CS sends all the content received from each $LMSE_i$ to the upper-level CS. Transmission of content terminates when it reaches the top-level CS (that is, the LS).
   c. The LS calculates the value of $idf(k) = \log(\sum N_i / \sum n_{k,i})$ from $N_{k,i}$ and N_i for each word k.
4. Update phase 2: Each $LMSE_i$ is updated as follows.
   a. $LMSE_i$ receives the set of Boolean queries Q which has been searched and the set of *idf* values from the LS.
   b. The Engine I/F of the $LMSE_i$ obtains from the corresponding LSE the highest score $\max_{d \in D} S_i(d,q)$ for each $q \in \{Q, K_i\}$, where $S_i(d,k)$ is the score of document d containing k, and D is the set of all the documents at the site, and sends all of these to the CS together with its own URL.
   c. The CS sends the content received from each $LMSE_i$ to the upper-level CS. Transmission of content terminates when it reaches the top-level CS (that is, the LS).

Note that the data transferred between each module are mainly used in the distributed calculation to obtain the score based on the *tf*idf* method. We call this method the distributed *tf*idf* method. The score based on the distributed *tf*idf* method is calculated during the search process. Thus, we provide further details about the score when we explain the search process in the CSE.

To achieve good performance in the update process, we sacrifice performance in the search process in the CSE, which is carried out as follows.

1. When $LMSE_0$ receives a query from a user, it sends the query to the CS.
2. The CS obtains from the LS all the LMSEs expected to have documents satisfying the query.
3. The CS sends the query to each of the LMSEs obtained.
4. Each LMSE searches for documents satisfying the query using the LSE, and returns the results to the CS.
5. The CS combines all the results received from LMSEs, and returns these to $LMSE_0$.
6. $LMSE_0$ displays the search results to the user.

Next, we describe the design of the scalable architecture for the distributed search engine, CSE.

During searching in the CSE, a problem arises in that communication delays occur. This problem is solved by the following techniques.

- Look Ahead Cache in "Next 10" Search [4]: To shorten the delay in the search process, the CS prepares the next result for the "Next 10" search. That is, the search result is divided into page units, and each page unit is cached in advance by a background process without increasing the response time.
- Score based Site Selection (SbSS) [5]: In a "Next 10" search, the score of the next ranked document in each site is gathered in advance, and requests to sites with low-ranked documents are suppressed. As a result of this suppression, network

traffic does not increase unnecessarily. For example, there are more than 100,000 domain sites in Japan. However, by using this technique, about ten sites are sufficient to satisfy requests from each continuous search.

- Global Shared Cache (GSC) [6]: An LMSE sends a query to the nearest CS. Several CSs may send the same requests to the LMSEs. Therefore, to share cached retrieval results globally among CSs, we proposed a GSC. In this method, the LS remembers the authority CS_a of each query and informs the CS_a of the CSs instead of the LMSEs. The CS caches the cached contents of CS_a.
- Persistent Cache (PC) [7]: There must be at least one CS in the CSE to improve the response time of retrieval. However, the cache soon becomes invalid because the update interval is very short in the CSE. The valuable first page is also lost. Therefore, we need a persistent cache that holds valid cache data before and after updating. In this method, there are two update phases. In the first update phase, each LMSE sends the number of documents that include each word to the LS, which detects the *idf* of each word. In the second update phase, a preliminary search is performed using the new *idf*s to update the caches.
- Query based Site Selection (QbSS) [8]: The CSE supports a Boolean search based on Boolean formula. In the Boolean search, the operations “and”, “or”, and “and-not” are available. Let S_A and S_B be the set of target sites for search queries A and B, respectively. Then, the set of target sites for queries “A and B”, “A or B”, and “A and-not B” are $S_A \cap S_B$, $S_A \cup S_B$, and S_A, respectively. Using this selection of target sites, the number of messages in the search process is saved.

These techniques are used as follows:

```
if the previous page of “Next 10” search has already been searched
  LAC
else if query does not contain “and” or “and-not”
  SbSS
else if it has been searched since index was updated
  GSC
else if it has been searched once
  PC
else // query is new
  QbSS
fi
```

2.2 Cloud Computing

Generally, cloud systems employ scale-out technology. In scale-out, many reasonable servers are used instead of the expensive high performance servers used in scale-up. Furthermore, in a cloud, virtualization is usually used. Virtualization techniques allow more than one virtual machine to be allocated to a physical machine. As a result, the utilization of machine resources is very high, thus reducing the cost.

According to NIST 1, cloud systems are classified according to the following types: SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS

(Infrastructure as a Service). In this paper, we focus on IaaS clouds such as AWS (Amazon Web Service). Recently, many cloud service providers have been established. They often provide IaaS because this is suited to the efficient management of data centers. Almost all cloud service providers own data centers as their core business and they can reduce the administration cost using cloud technology.

AWS is the most famous and popular IaaS. IaaS aims to provide a virtually organized computing environment. In IaaS, a user can use the required resources quickly on demand. This reflects the elasticity. For instance, the elastic IaaS part of the AWS is called EC2 (Elastic Compute Cloud). The Amazon EC2 provides functions for creating an instance of a virtual machine, and for starting/stopping it. When a virtual machine terminates, data stored in the machine is also deleted. In this sense, the HDD of a virtual machine is similar to RAM in a physical machine. Using Amazon EC2, a user can obtain as many servers as he likes. This feature is most suited to small and medium sized businesses. In AWS, if you need persistent data, you have to use another service such as S3 (Simple Storage Service) or EBS (Elastic Block Store). A user stores persistent data, which could be important, in such storage. Such storage is the target of our cloud search engine. In this paper, we propose a cloud search engine suitable for searching documents in EBS, which is used by being mounted in an instance. Therefore, from the viewpoint of a cloud search engine, it is viewed as a conventional file system.

In such a cloud, a user often increases the number of servers if the system performance/capacity is not adequate. At that time, resource allocation issues in the cloud need to be dealt with. One of the solutions is consistent hashing, which is a popular technique in overlay networks such as P2P (Peer to Peer). A typical consistent hashing is DHT (Distributed Hash Table), which is used in Chord. In Chord, peers are networked as a ring. For example, the order of peers in the network can be decided by their ID. In consistent hashing, the hash value of the contents is used as the ID. Each peer has a shortcut table in log order. In this way, in Chord, a message can be transferred within O(log n).

3 Cloud Search Engine

Here we propose a cloud search engine based on the CSE that can search documents in an IaaS cloud. The CSE is suited to an IaaS cloud such as Amazon EC2, especially where documents are accessed as conventional files. However, there are several issues if normal CSE is applied to such a cloud.

One of the issues is reliability, especially fault tolerance. In the LSE, which is a component of the CSE, there is no redundancy. For example, index data are not removed even if the server stops, because it is stored in a physical server. However, in an IaaS cloud, when an instance, which is a virtual machine, terminates, the index data are removed. Therefore, redundancy such as replication is required to realize a cloud search engine for an IaaS cloud.

Another issue is the scale-out technique. How to scale out is very important for an IaaS cloud. In conventional CSE, there is only one LS. Therefore, it is hard to scale out because of this bottleneck. In a previous work 10, we proposed a redundant architecture with multiple LSs for the CSE to solve this issue. In this paper, we propose another solution based on NoSQL technology, which is popular in clouds. A conventional SQL based DBMS is a bottleneck in a cloud. Thus, to remove this bottleneck, cloud based systems employ a memory based KVS (Key Value Store), which is simpler than a SQL based DBMS. Such a system is called NoSQL. We employ Apache Cassandra 12 as the NoSQL. Furthermore, in this paper, we propose redundant configurations for the LSEs based on consistent hashing. This configuration enables the communication traffic to be reduced by reallocating data when a new node joins.

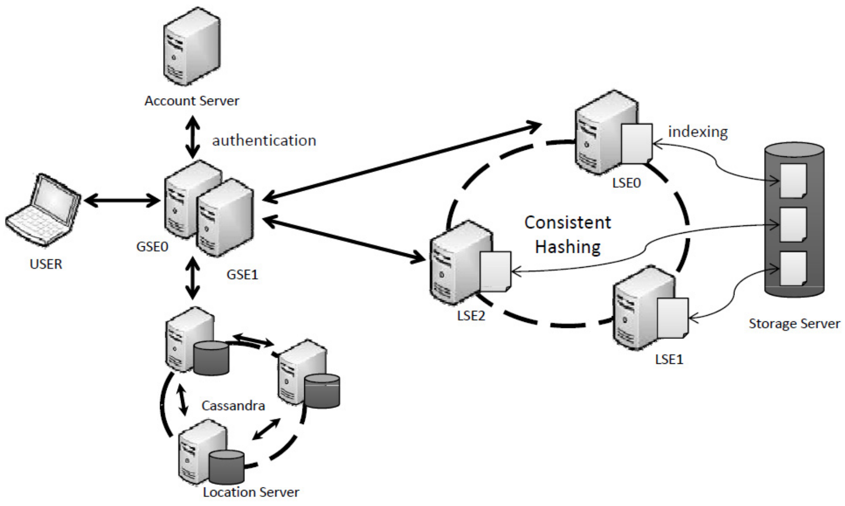


Fig. 2. The overview of Cloud Search Engine

A system overview of the proposed cloud search engine is illustrated in Figure 2. The following 5 components are shown in this figure.

- **Local Search Engine (LSE):** The LSE is a local search engine that searches local documents at each site. In our cloud search engine, more than one LSE organizes a group that shares documents. In a group, the index is replicated to each LSE. In this way, fault tolerance is realized in the LSE.
- **Global Search Engine (GSE):** The GSE accepts a user query and delegates it to the LSEs that are selected by the LS. Then, it shows the results in rank order after the ranking is modified to a global order. Many GSEs are allowed in a system, which ensures that the GSE loads are balanced.
- **Account Server (AS):** The AS manages account information, which is used for access control of documents. Generally, in a cloud, applications have multi-tenancy. This often means that documents owned by more than one organization

are mixed in storage. These documents are indexed by the same LSE. Therefore, access control is essential in a cloud search.

- **Location Server (LS):** When a user searches documents with the given keywords, the LS selects a set of LSEs with the keywords. Therefore, the LS is also called a meta search server. In addition, the LS stores meta knowledge, what each LSE knows, which is actually the keywords list stored in each LSE's index. In a cloud search engine, multiple LSs are included to increase reliability.
- **Storage Server (SS):** SS is a server that stores the searched documents. Generally, in an IaaS cloud, when an instance terminates, all of its data are removed even if the data are stored on a virtual HDD. Therefore, persistent data should be stored in reliable storage. The SS realizes storage like the EBS. In this implementation, nodes share documents with the SS by NFS.

Here, we describe an implementation of our proposed cloud search engine. First, we focus on creating indexes. In our cloud search engine, each LSE creates an index of local documents. We employ Apache Solr 11 for the LSEs. Apache Solr is a popular search engine that has many functions, such as facet search, highlighting hit keywords, and replication. In addition, we can develop additional functions freely in Solr because it is OSS.

The original CSE focused on fresh information retrieval. Therefore, the new cloud search engine also focuses on this. To realize fresh information retrieval, we have developed LSEDaemon, which observes specified directories. LSEDaemon runs with the LSE. When a file is updated, LSEDaemon notifies the LSE, which then updates the index. If the file is newly created, LSEDaemon decides which LSE owns it based on the document ID. However, this may cause the load to become unbalanced. Therefore, we introduce about 100 virtual nodes corresponding to physical nodes. Furthermore, LSEDaemon creates an XML configuration file that specifies the replication method, observes whether a node is dead or alive, and sends a summary of the index to the LS if the index is updated.

The LS is the most important component because it manages meta information used for site selection. It is referenced by all the GSEs and as such is the single point of failure and the bottleneck for performance. Thus, it is recommended that more than one LS is used. We implement the LS using Apache Cassandra, which can realize a reliable and high performance KVS(Key-Value Store).

Here, we describe the behavior for updating documents. The process given below is started by each LSE independently.

1. LSEDaemon observes a specified directory. When a file in the directory is modified, LSEDaemon computes the consistent hashing value from the document ID, assigns the document to an LSE, and creates the index.
2. After the index is updated, LSEDaemon sends a summary of the index to the LS, including the total number of documents in the LSE, the total list of keywords, and the number of documents that contain a keyword.

Next, we explain the behavior for searching.

1. A user requests a query from the GSE. The query includes the account information of the user. The GSE delegates the AS to authenticate the user and also manages the session, which is used for the "next 10" search.
2. The GSE asks the LS which LSEs know the keywords. The LS responds with the addresses of the selected LSEs.
3. The GSE sends the user's query to the LSEs selected by the LS.
4. Each LSE answers the query.
5. The GSE collects the search results and sorts them by rank order.
6. The GSE shows the relative top 10 items as an HTML document. The remaining search results are cached in the GSE for the "next 10" search.

One of the issues in a distributed search engine is consistent ranking. Generally, the ranking in a global search is different from the ranking in a local search. For example, in the *tf*idf* ranking method that is used in Apache Solr, the ranking of a document is decided by both *tf* and *idf*. Here, *tf* is the term frequency, and *idf* is the inverse document factor. Although *tf* is computed locally, *idf* should be computed globally. Therefore, we have to modify the ranking of results searched by Solr.

In Solr, the ranking is decided by the score of each document with respect to a query. The score S is given by the following equations.

$$\mathrm{S(q,d) = crd(q,d) \cdot nrm_{query}(q) \sum_{t\ in\ q}\{tf(t) \cdot idf(t)^2 \cdot bst(t) \cdot nrm(t,d)\}} \quad (1)$$

$$\mathrm{nrm_{query}(q)} = \frac{1}{\sqrt{\mathrm{ssw(q)}}} \quad (2)$$

$$\mathrm{ssw(q) = bst(q)^2 \cdot \sum_{t\ in\ q}\{idf(t) \cdot bst(t)\}^2} \quad (3)$$

$$\mathrm{idf(t)} = 1 + \log\left(\frac{\mathrm{maxDocs}}{1+\mathrm{docFreq}}\right) \quad (4)$$

Here w, t are terms, q is a query, d is a document, maxDocs is the maximum number of documents, and docFreq is the document frequency. In the CSE, the *idf* value is globally correct because the LS knows both maxDocs and docFreq. GSE resort search results which are collected from a set of LSEs in global order. Furthermore, *idf* is used to define the equation (3). In the above equations, *idf* is used with several boost values. These boost values can be specified by the application program.

Next, we discuss the fault tolerance of our system.

An LS is realized using Apache Cassandra. Meta data of an LS are automatically replicated to many other LSs. Therefore, meta data can be read from a set of LSs even if some of the LSs terminate.

An LSE is realized using Apache Solr. The LSE managing an index is decided by the document IDs in the index. Each LSE has a forwarding link to replicated data. Therefore, if an LSE terminates, a client can get the next LSE by following the forwarding link of the data.

The GSE is originally developed. However, the GSE is based on a zero share architecture, except for caching. A user freely selects any GSE. Therefore, if at least one GSE is running, the user's request can be accepted by a GSE.

4 Evaluations

In this paper, we evaluate our implementation of a cloud search engine on a PC. All components of the cloud search engine, the AS, LS, GSE, and LSEs (0-2) are deployed on this PC. However, the LSEs run in virtual machines as node0 and node1, on VMware Fusion. This configuration is shown in Table 1. Furthermore, we use the literature of Aozora Bunko 13 as the searched documents. The total number of documents is 1212, divided into 3 folders. Each folder has 404 documents. These folders are mounted in the LSEs using a shared folder of VMware Fusion. This configuration is shown in Table 2.

Table 1. Specification of the testbed

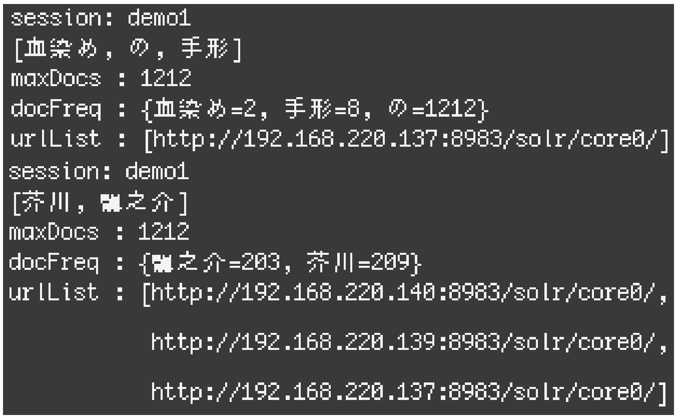
Specification type	Physical Machine (PM)	Virtual Machine (VM)
CPU	Intel Core 2 Duo 2 GHz	1 core
Memory	4GB	768MB
OS	Mac OS X	Ubuntu Server 9.04

Table 2. Number of documents stored in each node

LSE	*Node$_0$*	*Node$_1$ (replica)*
0	400	411
1	401	400
2	411	401

First, we evaluate index creation by prototyping. In this evaluation, we deploy 3 sets of documents to 3 LSEs (see Table 2). In Table 2, node$_0$ is the master of replication, while node$_1$ is the slave of replication. The order of consistent hashing is as follows: LSE0, LSE1, and LSE2. This means that LSE1 and LSE2 are slaves of LSE0 and LSE1, respectively. Therefore, if any LSE fails, its replica can respond to its requests.

We also evaluated the search results of the cloud search engine. Figure 3 shows the results when a user searches using keywords such as "血染めの手形"(bloodstained notes) the book title written by Mitsuzo Sasaki, and "芥川龍之介"(Ryunosuke Akutagawa) who is the author of Rashomon. Here, we employ Japanese keywords to test the Japanese full text search. In Figure 3, the number of LSEs that replied to the query "bloodstained notes" and "Ryunosuke Akutagawa" was 1 and 3, respectively. In particular, in the case of "Akutagawa Ryunosuke", the load in the cloud search engine can be balanced by using more than one LSE.

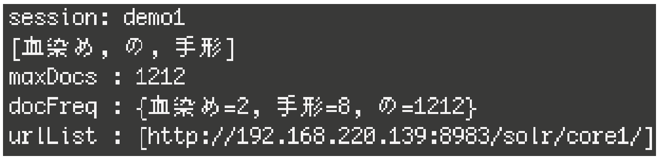


```
session: demo1
[血染め, の, 手形]
maxDocs : 1212
docFreq : {血染め=2, 手形=8, の=1212}
urlList : [http://192.168.220.137:8983/solr/core0/]
session: demo1
[芥川, 龍之介]
maxDocs : 1212
docFreq : {龍之介=203, 芥川=209}
urlList : [http://192.168.220.140:8983/solr/core0/,

          http://192.168.220.139:8983/solr/core0/,

          http://192.168.220.137:8983/solr/core0/]
```

Fig. 3. Search Results



```
session: demo1
[血染め, の, 手形]
maxDocs : 1212
docFreq : {血染め=2, 手形=8, の=1212}
urlList : [http://192.168.220.139:8983/solr/core1/]
```

Fig. 4. Search Results in the event that LSE0 stops

Here, we evaluate the search results in the event that any LSE terminates. In Figure 4, LSE0 stops. In this case, the cloud search engine replies to 192.168.220.139 (LSE1) the forwarding link of 192.168.220.137 (LSE0).

Table 3. Number of documents before the configuration is changed

LSE	Node_0	Node_1 (replica)
0	445	363
1	363	445

Table 4. Number of documents after the configuration is changed

LSE	Node_0	Node_1 (replica)
0	580	138
1	494	580
2	138	494

Next, we describe the evaluation results when an LSE joins dynamically. This evaluation is different from the previous one for the following reason. In this evaluation, after the system has initially been created with LSE0 and LSE1, LSE2 is added to the system. As a result, the documents stored in LSE2 are distributed to the others and the index and forwarding link for replication are updated.

Table 3 shows the number of documents initially assigned to LSE0 and LSE1. At this time, LSE0 and LSE1 are paired with each other. Next, LSE2 is added. Table 4 shows the number of documents assigned to LSE0, LSE1, and LSE2 after the documents have been re-allocated. At this time, the configuration of the replication is the same as in Table 2. From these results, we conclude that the index and the forwarding links for replication are correctly updated after the LSE is added. This means that it is possible to scale out this cloud search engine. However, the number of documents in each LSE is not balanced. This remains one of our future works.

5 Conclusions

In this paper, we presented a cloud search engine for an IaaS cloud, which is constructed by integrating multiple local search engines. To increase reliability, we employ Apache Solr, which supports replication, in the form of LSEs. Using Apache Solr for an LSE, the index is redundant. Therefore, if an LSE terminates, documents can still be searched successfully. Furthermore, the performance can be scaled out.

However, we have not yet developed the re-allocation of index data when an LSE is added. Therefore, the index data become unbalanced. This is a future work. In addition, this time, we evaluated the prototype system in a local environment. Therefore, the number of LSs, implemented with Apache Cassandra, was only one. We aim to evaluate another prototype extended with more than one LS.

References

1. NIST: Cloud Computing, `http://csrc.nist.gov/groups/SNS/cloud-computing/documents/forumworkshop-may2010/nist_cloud_computing_forum-leaf.pdf`
2. Weider, C., Fullton, J., Spero, S.: Architecture of the Whois++ Index Service, RFC1913
3. The Namazu Project, Namazu, `http://www.namazu.org/`
4. Sato, N., Yamamoto, T., Nishida, Y., Uehara, M., Mori, H.: Look Ahead Cache for Next 10 in Cooperative Search Engine. In: Proc. of DPSWS 2000, vol. 2000(15), pp. 205–210 (December 2000) (in Japanese)
5. Sato, N., Uehara, M., Sakai, Y., Mori, H.: Score Based Site Selection in Cooperative Search Engine. In: Proc. of DICOMO 2001, vol. 2001(7), pp. 465–470 (June 2001) (in Japanese)
6. Sato, N., Uehara, M., Sakai, Y., Mori, H.: Global Shared Cache in Cooperative Search Engine. In: Proc. of DPSWS 2001, vol. 2001(13), pp. 219–224 (October 2001) (in Japanese)
7. Sato, N., Uehara, M., Sakai, Y., Mori, H.: Persistent Cache in Cooperative Search Engine. In: Proc. of The 4th International Workshop on Multimedia Network Systems and Applications (MNSA 2002), in Conjunction with The 22nd International Conference on Distributed Computing Systems (ICDCS 22), pp. 182–187 (2002)
8. Sakai, Y., Sato, N., Uehara, M., Mori, H.: The Optimal Monotonization for Search Queries in Cooperative Search Engine. In: Proc. of DICOMO 2001. IPSJ Symposium Series, vol. 2001(7), pp. 453–458 (June 2001) (in Japanese)

9. Sato, N., Udagawa, M., Uehara, M., Sakai, Y., Mori, H.: Query based Site Selection for Distributed Search Engines. In: Proc. of The 6th International Workshop on Multimedia Network Systems and Applications (MNSA 2003), in Conjunction with the 23th International Conference on Distributed Computing Systems (ICDCS 2003), pp. 556–561 (2003)
10. Sato, N., Udagawa, M., Uehara, M., Sakai, Y., Mori, H.: Redundancy of Meta Search Servers in a Distributed Search Engine. In: Proceedings of 17th International Conference on Advanced Information Networking and Applications (AINA 2003), pp. 400–407 (2003)
11. Apache Solr, `http://lucene.apache.org/solr/`
12. Apache Cassandra, `http://cassandra.apache.org/`
13. Aozora Bunko, `http://www.aozora.gr.jp/`

Data Scheduling in Data Grids and Data Centers: A Short Taxonomy of Problems and Intelligent Resolution Techniques

Joanna Kołodziej[1] and Samee Ullah Khan[2]

[1] Institute of Computer Science
Cracow University of Technology, ul. Warszawska 24, 31-155 Cracow, Poland
jkolodziej@uck.pk.edu.pl
[2] NDSU-CIIT Green Computing and Communications Laboratory,
North Dakota State University,
ND 58108, USA
samee.khan@ndsu.edu

Abstract. Data-aware scheduling in today's large-scale heterogeneous environments has become a major research issue. Data Grids (DGs) and Data Centers arise quite naturally to support needs of scientific communities to share, access, process, and manage large data collections geographically distributed. Data scheduling, although similar in nature with grid scheduling, is given rise to the definition of a new family of optimization problems. New requirements such as data transmission, decoupling of data from processing, data replication, data access and security are to be added to the scheduling problem are the basis for the definition of a whole taxonomy of data scheduling problems. In this paper we briefly survey the state-of-the-art in the domain. We exemplify the model and methodology for the case of data-aware independent job scheduling in computational grid and present several heuristic resolution methods for the problem.

Keywords: Data Grid, Scheduling, Data Center, Expected Time to Transmit, Data replication.

1 Introduction

Traditional scheduling problems are mainly concerned with high performance parameters related to task processing (CPU related parameters) such as makespan, flowtime, resource usage, etc. These parameters usually do no take into account requirements on data needed for task completion such as data transmission time, data access rights, data availability (replication) and security issues. In most of the research on grid and cloud computing data transmission time is assured to be fast/very fast, data access rights are granted, due to the single domain of LANs and clusters, so there is no need for special data access management. Similarly, security issues are easily handled within the same administrative domain. Obviously, the situation is very different in current large scale setting, where data sources needed for task completion could be located at different sites under different administrative domains.

Although data-aware scheduling has been considered in a significant volume of the research works, e.g. in parameter sweep applications [2,3], the scheduling problems in Computational Grids (CGs) and in Data Grids (DGs) is dealing with in a separated way. Much of the current efforts are focused on scheduling workloads in a data center or schedule movement of data and data placement [38] for efficient resource/storage utilization or energy-effective scheduling in large-scale data centers [37], [8], [29], [18], [44]. A recent example is that of GridBatch [40] for large scale data-intensive problems on cloud infrastructures.

Due to advent of DGs and fast development of Cloud Computing, data-aware scheduling has recently attracted considerable attention of researcher from distributed computing and optimization communities. In fact, DGs can be seen as precursors of Data Centers in Cloud Computing platforms, which serve as basis for collaboration at a large scale. In such computational infrastructures, the large amount of data to be efficiently processed is a real challenge. One of the key issues contributing to the efficiency of massive processing is the scheduling with data transmission requirements.

In this work, we consider the data-aware scheduling aiming to problem formulations that take into account new requirements such as data transmission, decoupling of data from processing [32], [47], [53],, data replication [7], [10], data access and security, [33], [16], [17]. The aim is to integrate these new requirements into a multi-objective optimization model in a similar way that it has been addressed for a classical grid scheduling. The grid schedulers must thus take into account the features of both CG and of DG to achieve desired performance of grid-enabled applications [34], [35]. We exemplify the approach for the case of data-aware independent batch task scheduling problem.

The remainder of this paper is structured as follows. We present in Section 2 a high level taxonomy for data scheduling in Data Grids. The data-aware system model for independent batch scheduling is given in Section 3. Selected heuristic-based resolution methods for solving data-aware independent batch scheduling are presented briefly in Section 4. We discuss the most important challenges in data-aware scheduling in Section 5 and conclude this paper in Section 6.

2 A Short Taxonomy of Data-Aware Scheduling Problems in Data Grids

Data Grids (GGs) are defined as computational infrastructures that provide high performance massive aggregated computing resources and distributed data storage capabilities. DGs support data intensive applications. Among several types of Data Grids elements four components seem to be fundamental, namely *Grid Organization* module, *Data Replication* mechanism, *Data Transfer* policy and infrastructure and *Scheduling* module (see also [46]) as shown in Fig. 1.

The complex hierarchy of the DG can be then organized as a collection of four sub-hierarchies, each of them dedicated to one of the DG's elements. Such

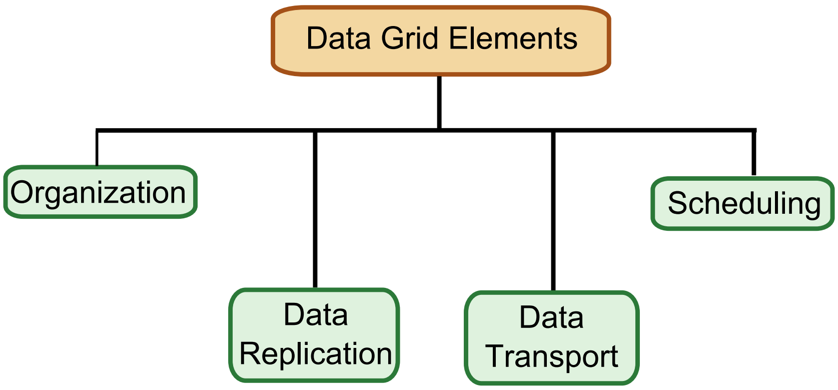


Fig. 1. Data Grid elements (based on the full taxonomy presented in [46])

complex DG characteristics are presented in [46]. In fact, each of the areas of data transport, replica management and resource management pose challenging research issues and can be analyzed as independent research areas. However, in the recent studies on grid systems there is a need to analyze the specific requirements of DG's users and environments, which in fact tends to a direct or indirect aggregation of the particular grid elements and methodologies into wider classes. In this paper we focus on the *Scheduling* sub-hierarchy.

The requirements for large data files and the presence of multiple replicas of these data files located at geographically-distributed data hosts makes scheduling of data-intensive tasks different from that of simply computational tasks. Schedulers have to take into account the network bandwidth availability and the latency of data transfer between a computational node to which a task is going to be submitted, and the storage resource(s) from which the data required is to be retrieved [19], [11]. Therefore, the scheduler needs to be aware of any replicas "close" to the computation node and if the replication is coupled to the scheduling, then create a new copy of the data.

A basic taxonomy for scheduling of data-intensive applications is shown in Fig. 2.

There are five main categories in the taxonomy, which can be characterized in the following paragraphs.

Application Model. Scheduling strategies in DGs can be classified by the application models, which are mainly determined by the manner in which the grid task is composed or distributed for scheduling. The grid tasks may be categorized into the following classes:

- *Process-oriented applications*– in this applications the data is manipulated at the process level (Message Passing Interface (MPI) programs [5]).
- *Independent tasks* – can have different objectives or may be defined as a meta-task or a bag-of-tasks. They are scheduled individually and it is ensured that each of them get the required share of resources for their completion [41].

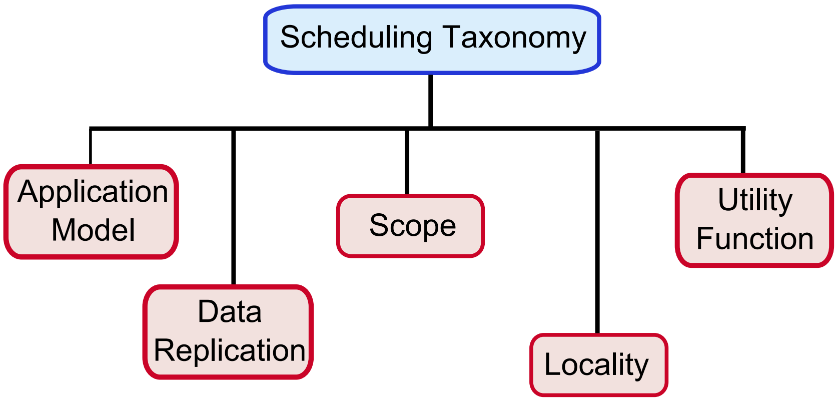


Fig. 2. Data Grid scheduling taxonomy

– *A workflow* ([45]) is a sequence of tasks in which each task is dependent on the results of its predecessor(-s) tasks. The outputs of the preceding tasks may be large data files themselves.

Scope. Scope relates to the modification of application of the scheduling strategy within the DG. If the scope is *individual*, then the scheduling strategy is concerned only with meeting the objectives from a user's perspective. In a multi-user environment, each scheduler would have its own independent view of the resources that it wants to utilize. A scheduler can be for example aware of fluctuations in resource availability, special security requirements and other policies set at the Virtual Organization level and enforced at the resource level ([48]).

Data Replication. This category relates to whether task scheduling is coupled to data replication or not. A comparison analysis of decoupled *vs.* coupled strategies performed in [42] has shown that decoupled strategies promise increased performance and reduce the complexity of designing algorithms for DG environments. Additionally, one can consider replicating full data sets or chunks of data sets. Related to data replication, there are usually a set of multiple user *QoS*. For instance, access time to data, data availability, etc. can be seen as QoS requirements. Certainly, such requirements should be taken into account by the grid scheduler [28], [9], [12], [13].

Utility Function. The utility function can vary depending on the requirements of the users and architecture of the distributed system that the algorithm is targeted to. Traditionally, scheduling algorithms have aimed at reducing in average the total time required for computing all the tasks in a given batch or set. Load balancing algorithms try to distribute load among the machines so that maximum work can be obtained out of the systems. Scheduling algorithms with economic objectives try to maximize the users' economic utility usually expressed as some profit function that takes into account economic costs of executing the jobs on the DG. Recently, one of the key objective in green grid, cloud and high performance computing centers is to optimize the energy utilization by the system.

Locality. Exploiting the locality of data has been a common technique for scheduling and load-balancing in parallel programs [6] and in query processing in databases [43]. Similarly, data grid scheduling algorithms can be categorized as whether they exploit the spatial or temporal locality of the data requests. Spatial locality is locating a task in such a way that all the data required for the task is available on data hosts that are located "close" to the node of computation. Temporal locality exploits the fact that if data required for a task is close to a compute node, subsequent tasks which require the same data are scheduled to the same node to benefit from the data proximity.

3 Data-Aware System Model for Independent Job Scheduling

Let us consider now the problem of batch scheduling of independent data-intensive applications onto computational grid's resources. The applications can be considered as meta-tasks, which require multiple data sets from different heterogeneous data hosts. These data sets may be replicated at various locations and can be transferred to the computational grid through the networks of various capabilities. A possible variant of this scenario is presented in Fig. 3.

Formally, the main components of the data-aware grid system can be characterized as follows:

- a meta-task $N_{batch} = \{t_1, \ldots, t_n\}$ defined as a batch of independent tasks, each of which can be executed just at one or more grid resources (n - is a total number of tasks in the batch);
- a set of computing grid nodes $M_{batch} = \{m_1, \ldots, m_k\}$, ($k$ - is a total number of machines available in the system for a given batch;
- a set of data-files $F_{batch} = \{f_1, \ldots, f_r\}$ needed for the batch execution;
- a set of data-hosts $DH = \{dh_1, \ldots, dh_s\}$ dedicated for the data storage purposes, having the necessary data services capabilities.

3.1 Task Workload and Computing Capacities

The computational load of the batch can be defined as a *tasks workload vector* $WL_{batch} = [wl_1, \ldots, wl_n]$, where wl_j denotes an estimation of the computational load of a task t_j (in Millions of Instructions –MI). Each task t_j requires a set of files $F_j = \{f_{(1,j)}, \ldots, f_{(r,j)}\}$ ($F_j \subseteq F_{batch}$) that are distributed on a subset DH_j of DH. Specifically, for each file $f_{(p,j)} \in F_j$ ($p \in \{1, \ldots, r\}$), DH_j is the set of data hosts, on which $f_{(p,j)}$ is replicated, and from which it is available. We assume that each data host can serve multiple data files at a time and data

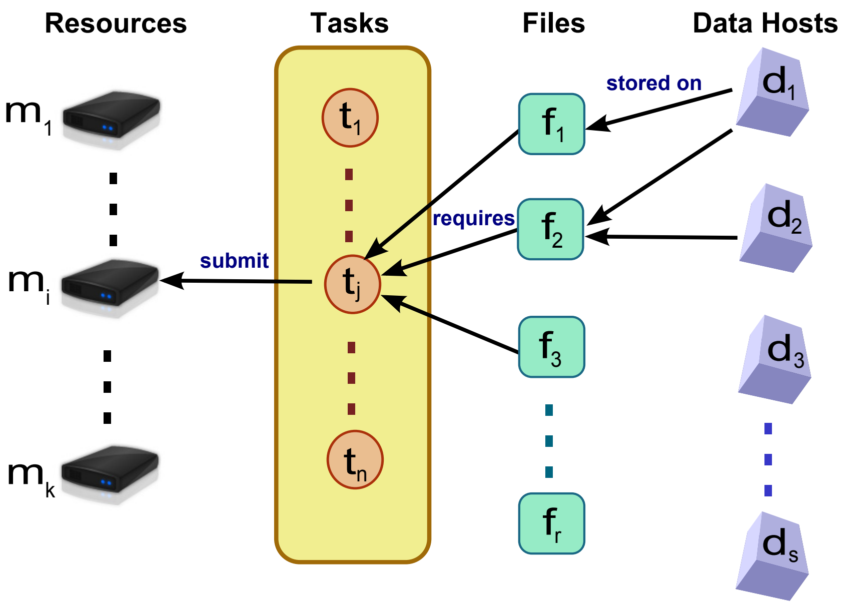


Fig. 3. Data-aware meta-task grid scheduling problem

replication is *a priori* defined as a separate replication process that may take into consideration various factors such as locality of access, load on the data-host and available storage space.

The computational power in the grid system can be characterized by its processing speed expressed by a clock frequency or by its computing capacity specified in MIPS (Million Instructions Per Second). The computing capacity of the resources available for processing a given batch is defined by a *computing capacity vector* $CC_{batch} = [cc_1, \ldots, cc_m]$, in which cc_i denotes the computing capacity of machine i. The estimation of the prior load of each machine from M_{batch} can be represented by a *ready times vector* $ready\_times_{(batch)} = [ready_1, \ldots, ready_m]$.

3.2 Data-Aware Task Execution Time Model

The data-aware task execution time model presented here follows an *Expected Time to Compute (ETC)* matrix model [1], in which an ETC matrix is defined, $ETC = [ETC[j][i]]_{n \times m}$ where $ETC[j][i]$ is an expected (estimated) time needed for the computing the task t_j on machine m_i. The workload and computing capacity parameters for tasks and machines are generated by using the Gamma probability distribution for the expression of tasks and machines heterogeneities in the grid system. The elements of the ETC matrix can be computed as the ratio of the coordinates of WL and CC vectors, i.e.:

$$ETC[j][i] = \frac{wl_j}{cc_i}. \tag{1}$$

The expected execution time of the task t_j in Eq. (1) depends on the processing speed of the machine m_i. However, for the successful task execution we need to include in the model the time needed for data transfer. For each data file $f_{(p,j)} \in F_j$ ($p \in \{1, \ldots, r\}$), the time required to transfer $f_{(p,j)}$ from the data host $dh_{(p,j)} \in D_j$ to i, denoted TT, is defined by the following formulae:

$$TT[i][j][f_{(p,j)}] = response\_time(dh_{(p,j)}) + \frac{Size\left[f_{(p,j)}\right]}{B(dh_{(p,j)}, i)} \tag{2}$$

where $response\_time(dh_{(p,j)})$ is the difference between the time when the request was made to $dh_{(p,j)}$ and the time when the first byte of the data file $f_{(p,j)}$ is received at machine m_i. This is an increasing function of the load on the data host. We denote by $B(dh_{(p,j)}, i)$ in Eq. (2) the bandwidth of the logical link between $dh_{(p,j)}$ and m_i[1] The estimated completion time for the task t_j on machine m_i, $completion[j][i]$, is the wall-clock time taken for the task from its submission till completion and is a function of computing and transmission times specified in Eq. (1) and (2). The impact of the data transfer time on the task completion time depends on the mode, in which the data files are processed by the task. The are two main such scenarios presented briefly in Fig. 4 (see also [45]).

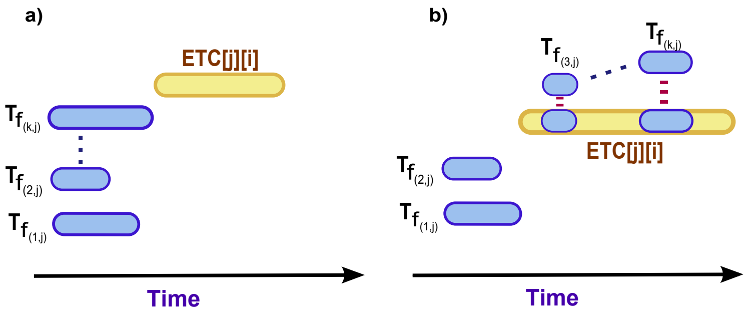


Fig. 4. Two variants of estimation of completion time of task t_j on machine m_i with the assumption of k data files needed for the task execution

In Fig. 4, for convenience, we denote the times for transferring the files $f_{(1,j)}, f_{(2,j)}, \ldots, f_{(k,j)}$ by $T_{f_{(1,j)}}, T_{f_{(2,j)}}, \ldots, T_{f_{(k,j)}}$, respectively. In the first scenario presented in Fig. 4(a) the data files needed for the task execution are transferred in parallel *before* the task execution. The number of simultaneous

[1] The physical network between the data hosts and resources consists of several entities such as routers, switches, links and hubs. However, the model presented here abstracts the physical network to consider just a logical network topology where each machine is connected to every data host by a distinct network link. Thus the bandwidth of the logical link between data host and machine is the bottleneck bandwidth of the actual physical network between them.

data transfers determines the bandwidth available for each transfer. Thus, the time of completion of the task t_j on machine m_i can be calculated by using the following formulae:

$$completion[j][i] = \max_{f_{(p,j)} \in F_j} TT[i][j][f_{(p,j)}] + ETC[j][i]. \tag{3}$$

On the other hand, Fig. 4(b) represents the idea of the second scenario, in which some of the data files are transferred completely prior to the task execution and the rest are accessed as streams during the execution. In this case, the transfer times of the streamed data files are masked by the computation time of the task, and, in the result, increase this computation time. The completion time of the task t_j on machine m_i can be calculated in the following way:

$$\begin{aligned} completion[j][i] = \max\nolimits_{f_{(p,j)} \in \widehat{F_j}} TT[i][j][f_{(p,j)}] + \\ + \sum\nolimits_{f_{(l,j)} \in [F_j \setminus \widehat{F_j}]} TT[i][j][f_{(l,j)}] ETC[j][i]. \end{aligned} \tag{4}$$

where $\widehat{F_j}$ denotes a set of data files which are transferred prior the task execution.

We consider the data hosts as the data storage centers separated from the computing resources in order to make the system adaptable to various scheduling scenarios. Of course, in particular cases we can assume that each computing resource has its own data storage module [14], [20]. In such a case the internal data transfer times should be rather low and can be omitted in the analysis. However, for a fair estimation of the data transfers from the other computing sources there is a need in fact to decouple the data storage module from the computing module in the resource architecture [15], [22], [24]. The scalability and effectiveness of the whole system depends strongly on the replication mechanism and the resource data storage and computation capacities, [30], [23], [25], [21], which in some cases can be the main barrier in the schedulers' performance improvement [26], [27].

3.3 Scheduling Phases and Objectives

Scheduling phases in the data-ware scheduling are similar to Grid scheduling without data sets, but now it is assumed that Grid information services include also services for replicas such as replica management, discovery besides file transfer capabilities. These phases can be resumed as follows:

1. Get the information on available resources ;
2. Get the information on pending tasks ;
3. Get the information on data hosts where data files for tasks completion are required;
4. Prepare a batch of tasks and compute a schedule for that batch on available machines and data hosts;
5. Allocate tasks;
6. Monitor (failed tasks are re-scheduled).

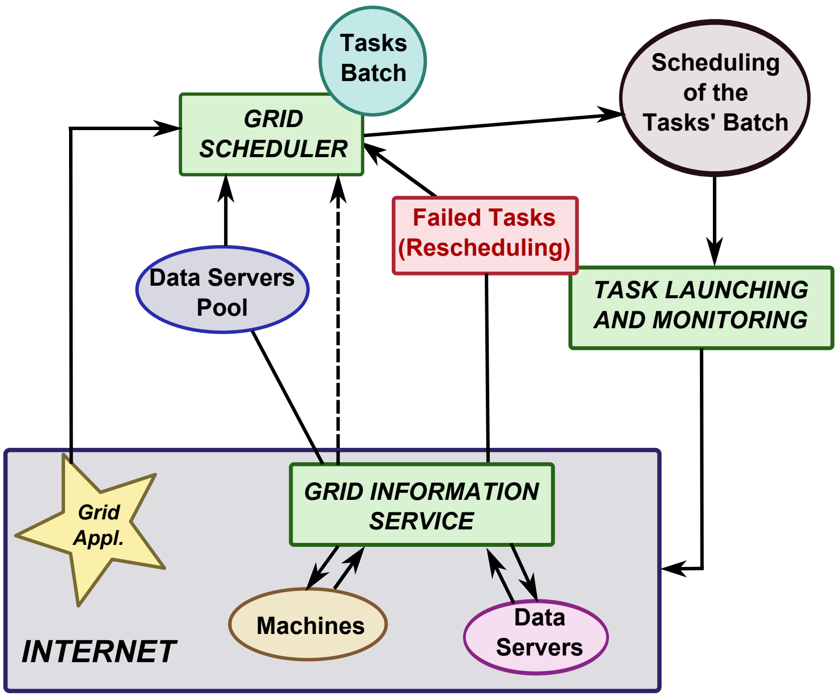


Fig. 5. Phases of the data-aware batch scheduler

These steps can be graphically represented as in Fig. 5.

The main objectives in data-aware scheduling are similar to the objectives formulated for grid scheduling without data files and include minimization of completion time, makespan, flowtime, etc.:

– Minimizing completion time of the task batch:

$$\sum_{t_j \in N_{batch}; m_i \in M_{batch}} completion[t_j][m_i]$$

where $completion[t_j][m_i]$ is defined as in Eq. 4).

– Minimizing makespan:

$$\min_{Sched} \max_{m_i \in M_{batch}} completion[m_i]$$

where $completion[m_i]$ is computed as the sum of completion times of tasks assigned to machine m_i.

Additionally, there are objectives related to the data-aware nature of the scheduling, such as access cost, response time, optimized QoS, etc. For example access cost an be computed as a weighted sum of reading cost and writing cost. Minimizing access cost affects directly the task turnaround time.

3.4 Strategies for Enhancing Data-Aware Schedulers

Several techniques can be used to reduce the transmission and access time in data-aware scheduling. As mentioned earlier, replication is a primary technique in this regard, which increases data availability, and therefore, increases scheduler's reliability. Another useful technique is that of parallel downloading of replicated data. Due to the dynamics of Grid systems, instead of replicating full data files, chunks of data files are replicated, which can further downloaded in parallel from different data hosts (see e.g. [52]).

In a similar vein, techniques used in P2P networks for downloading files can be used within the data-aware scheduling framework. The idea is that we could defined a virtual overlay on top of the Grid system by defining neighboring relations among computing sites and data hosts if computing sites contain replicas of data fragments for execution of a task assigned to the computing site. Then, we can formulate an optimization problem consisting in finding a subset of peer neighbors of the computing site from where to download/receive the data fragments [39]. The problem can be formally defined as follows.

Definition 1 (Neighbor-selection problem). *A neighbor-selection problem in P2P networks problem can be defined as* $\prod = (N, C, M, F, s)$*, in which* N *is the number of peers,* C *is the entire collection of content fragments,* M *is the maximum number of the available online peers,* F *is a single objective to optimize the number of swap fragments, or multi-objective to optimize the number of swap fragments, and to minimize the downloading cost;* s *denotes the environment constraints. The key components are operations, machines and data-hosts.*

The near-optimal resolution of this problem [39] can be used at the scheduling phase of selecting data hosts from where to get the data need for completion of the tasks in the batch.

4 Resolution Methods

4.1 Ad Hoc Methods

Ad hoc heuristics are simple procedures that need not to find even near-optimal solutions but are very fast and easy to implement. We briefly mention here some ad hoc heuristics for data-aware scheduling. An exhaustive list presented for Grid scheduling without data requirements can be found in [50] and [51].

MinMin Heuristic. In [45] the authors propose an extension of MinMin and Sufferage heuristics. In this extension they take into account the distributed data requirements of the target application model. The basic idea of the modified MinMin heuristic is to match in the beginning the meta-task to a resource set that guarantees the minimal completion time for some task in a batch. This is produced through special matching heuristics. They define Set Covering Problem (SCP) Tree Search (see also [4]), Greedy Selection, Compute-First or Exhaustive Search heuristics, which allow to select an appropriate combination of data hosts and a compute resource that the total completion time for a given component of meta-task is minimized[2]. Then, the task with the optimal (minimal) completion time in the present allocation is assigned to the compute resource. This task is then removed from the batch structure. As task assignment changes the availability of the compute resource with respect to the number of available slots/processors, the resource information is updated and the process is repeated until all the components of the meta-task have been allocated to some resource set. When a task is scheduled for execution on a compute resource, all required data files which are not available local to the resource, are transferred to the resource prior to execution. These data files become replicas that can be used by following meta-task components.

Sufferage Heuristic. The motivation behind the modified Sufferage heuristic is to allocate a resource set to a meta-task that would be disadvantaged the most (or "suffer" the most) if that resource set were not allocated to it. This is determined through a sufferage value computed as the difference between the second best and the best value of the completion time for the meta-task components. For each task, the resource that offers the least value of the completion time is determined through the same mechanisms as that in MinMin. Then another resource with the second minimal completion time is selected to establish the 'sufferage' value for a given task. The selection of the compute resource determines both the task execution time and the data transfer times. After determining the sufferage value for each task, the task with the largest sufferage value is then selected and assigned to its chosen resource. The rest of the heuristic including dispatching and updating of compute resource and data host information proceeds in the same manner as MinMin.

Other interesting ad hoc methods are Shortest Turnaround Time (STT), Least Relative Load (LRL) and Data Present (DP) (see e.g. [52]).

4.2 Meta-heuristic Methods

Dealing with the many constraints and optimization criteria in a dynamic environment scheduling of data-intensive applications in Computational Grid remains very complex and computationally hard problem [31], [36]. The significance of meta-heuristic approaches for designing efficient grid schedulers can be explained as follows (see also [49]:

Meta-heuristics Are Well Understood. Meta-heuristics have been studied for a large number of optimization problems, from theoretical, practical and experimental perspectives.

Computing Near-Optimal Solutions. In the dynamic grid environment, it is usually impossible to generate the optimal schedules. This is so due to the fact that grid schedulers run as long as the grid system exists and thus the

[2] Referred to as the Minimum Resource Set (MRS) problem.

performance is measured not only for particular applications but also in the long run. Therefore, in such situation meta-heuristics are among best candidates to cope with grid scheduling.

Dealing with Multi-objective Nature. Meta-heuristics have proven to efficiently solve the complex multi-objective optimization problems.

Well Designed for Periodic and Batch Scheduling. In the case of periodic scheduling the resource provisioning can be done with no strong time restrictions. This means that we can run meta-heuristic-based schedulers for longer execution times and significantly increase the quality of generated schedules. In batch scheduling, we could run the meta-heuristic-based scheduler for the time interval comprised within two successive batches activations.

Hybridization with other Approaches. Meta-heuristics can be easily hybridized with other approaches, which is useful to make grid schedulers better adapted to various grid scenarios, grid types, specific types of applications, etc.

Designing Robust Grid Schedulers. The dynamics of the grid environment directly impacts on the performance of the grid scheduler. A robustness in grid scheduling is a key issue in high-quality resource allocation in the case of frequent changes in the system's states.

The heuristic scheduling methods are usually classified into three main groups, namely calculus-based (greedy algorithms and ad-Hoc methods), stochastic (guided and non-guided methods) and enumerative methods (dynamic programming and branch-and-bound algorithm). The most popular and efficient methods in grid scheduling are ad-hoc, local search-based and population-based methods. A simple taxonomy of the heuristic schedulers is presented in Fig. 6.

Each of this scheduler can be adapted to the data grid scheduling by adding some extra tasks-data files matching procedures.

5 Scheduling Challenges

The data-aware scheduling in grid systems becomes even more challenging when we consider the following key challenges:

- **Scheduling Policy:** According to the selection of data-hosts and mapping of resources, the optimization criteria such makespan and flowtime may change significantly.
- **Storage Constraints:** Only limited storage capacity is available at resources. As the tasks get executed, the data produced should be either deleted or moved. Storage aware resource scheduling problem is a major area of research. Data providence should be associated with scheduling policy.
- **Replication Policy:** The availability of replicas of data and their locality heavily depends on the replication policy. For expressing the system dynamics a dynamic replication policy that can balance the replicas among data hosts should be formulated.

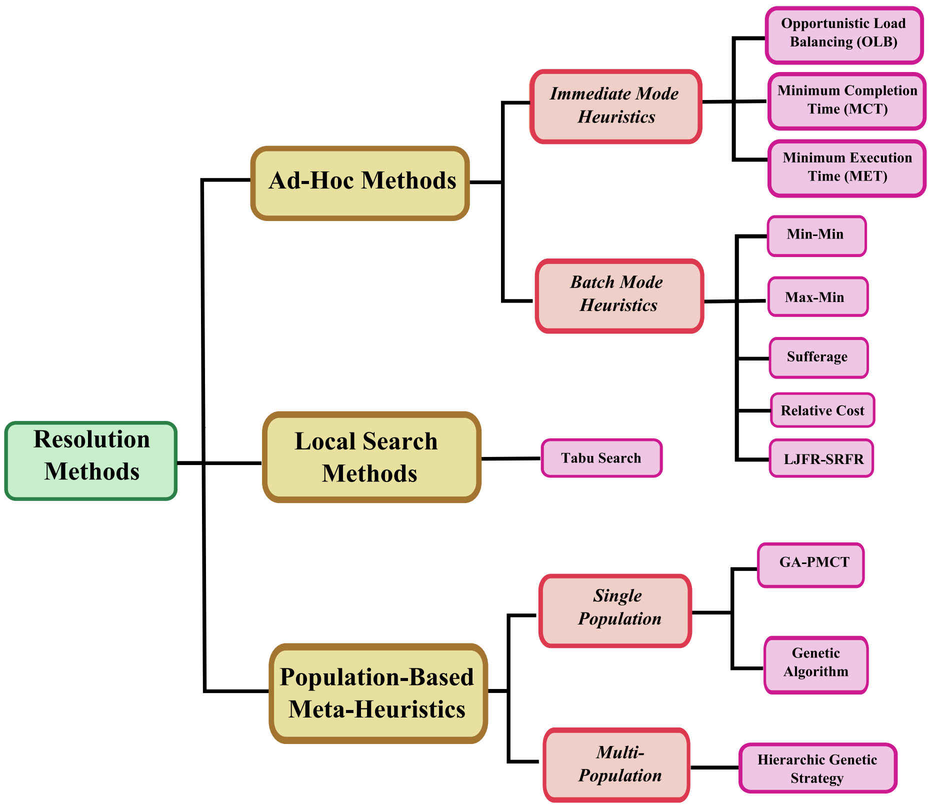


Fig. 6. Heuristic resolution methods taxonomy

– **Resource Provisioning:** In real life approaches there is a need to establish a Service-Level-Agreement (SLA) based advance reservation to circumvent the sudden scarcity of resources, which can guarantee the resource access at the scheduled time.
– **User QoS.** Users Quality of Service (QoS) criteria such as budget and deadline constraints may also be taken into account.
– **Security.** All operations in a Data Grid should be mediated by a security layer that handles authentication of entities and ensures conduct of only authorized operations. Additionally the grid users can specify their own criteria for secure task allocation at grid resources.

6 Conclusions and Future Work

In this paper we have presented a simple taxonomy of data-aware scheduling based on a set of requirements such data transmission, decoupling of data from processing, data replication and data access. By considering these requirements a family of data-aware scheduling problems can be defined, whose resolution can be very useful to design efficient data-aware schedulers. We have focused on the Data-aware Independent Batch Scheduling for which we have formalized the transmission time, in a way that it can be easily integrated into classical optimization objectives of grid scheduling. This is particularly useful as known optimization formulation and resolution methods can be applied to the data-aware scheduling with transmission times. We have also briefly discussed the different resolution methods (including ad hoc and meta-heuristics methods) to cope in practice with the complexities of the problem.

References

1. Ali, S., Siegel, H.J., Maheswaran, M., Hensgen, D.: Task execution time modeling for heterogeneous computing systems. In: Proceedings of Heterogeneous Computing Workshop, pp. 185–199 (2000)
2. Buyya, R., Murshed, M., Abramson, D., Venugopal, S.: Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm. Softw. Pract. Exper. 35(5), 491–512 (2005)
3. Casanova, H., Obertelli, G., Berman, F., Wolski, R.: The AppLeS parameter sweep template: user-level middleware for the grid. In: Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (CDROM) (Supercomputing 2000). IEEE Computer Society, Washington, DC (2000)
4. Christofides, N.: Independent and Dominating Sets–The Set Covering Problem. In: Graph Theory: An Algorithmic Approach, pp. 30–57 (1975) ISBN: 012 1743350 0
5. Foster, I., Karonis, N.: A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems. In: Proceedings of the IEEE/ACM SuperComputing Conference 1998 (SC 1998), San Jose, CA, USA, IEEE CS Press, Los Alamitos (1998)
6. Hockauf, R., Karl, W., Leberecht, M., Oberhuber, M., Wagner, M.: Exploiting Spatial and Temporal Locality of Accesses: A New Hardware-Based Monitoring Approach for DSM Systems. In: Pritchard, D., Reeve, J.S. (eds.) Euro-Par 1998. LNCS, vol. 1470, pp. 206–215. Springer, Heidelberg (1998)
7. Kliazovich, D., Bouvry, P., Khan, S.U.: DENS: Data Center Energy-Efficient Network-Aware Scheduling. In: ACM/IEEE International Conference on Green Computing and Communications (GreenCom), Hangzhou, China, pp. 69–75 (December 2010)
8. Kliazovich, D., Bouvry, P., Audzevich, Y., Khan, S.U.: GreenCloud: A Packet-level Simulator of Energy-aware Cloud Computing Data Centers. In: Proc. of the 53rd IEEE Global Communications Conference (Globecom), Miami, FL, USA (December 2010)
9. Khan, S.U., Ahmad, I.: A Pure Nash Equilibrium based Game Theoretical Method for Data Replication across Multiple Servers. IEEE Transactions on Knowledge and Data Engineering 21(4), 537–553 (2009)
10. Khan, S.U., Ardil, C.: A Weighted Sum Technique for the Joint Optimization of Performance and Power Consumption in Data Centers. International Journal of Electrical, Computer, and Systems Engineering 3(1), 35–40 (2009)

11. Khan, S.U.: A Multi-Objective Programming Approach for Resource Allocation in Data Centers. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NV, USA, pp. 152–158 (July 2009)
12. Khan, S.U.: On a Game Theoretical Methodology for Data Replication in Ad Hoc Networks. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NV, USA, pp. 232–238 (July 2009)
13. Khan, S.U.: A Frugal Auction Technique for Data Replication in Large Distributed Computing Systems. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NV, USA, pp. 17–23 (July 2009)
14. Khan, S.U., Ardil, C.: A Fast Replica Placement Methodology for Large-scale Distributed Computing Systems. In: International Conference on Parallel and Distributed Computing Systems (ICPDCS), Oslo, Norway, pp. 121–127 (July 2009)
15. Khan, S.U., Ardil, C.: A Competitive Replica Placement Methodology for Ad Hoc Networks. In: International Conference on Parallel and Distributed Computing Systems (ICPDCS), Oslo, Norway, pp. 128–133 (July 2009)
16. Khan, S.U., Ardil, C.: On the Joint Optimization of Performance and Power Consumption in Data Centers. In: International Conference on Distributed, High-Performance and Grid Computing (DHPGC), Singapore, pp. 660–666 (August 2009)
17. Khan, S.U.: A Self-adaptive Weighted Sum Technique for the Joint Optimization of Performance and Power Consumption in Data Centers. In: 22nd International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS), Louisville, KY, USA, pp. 13–18 (September 2009)
18. Khan, S.U.: A Goal Programming Approach for the Joint Optimization of Energy Consumption and Response Time in Computational Grids. In: Proc. of the 28th IEEE International Performance Computing and Communications Conference (IPCCC), Phoenix, AZ, USA, pp. 410–417 (December 2009)
19. Khan, S.U., Ahmad, I.: Non-cooperative, Semi-cooperative, and Cooperative Games-based Grid Resource Allocation. In: Proc. of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rhodes Island, Greece (April 2006)
20. Khan, S.U., Ahmad, I.: Comparison and Analysis of Ten Static Heuristics-based Internet Data Replication Techniques. Journal of Parallel and Distributed Computing 68(2), 113–136 (2008)
21. Khan, S.U., Ahmad, I.: Discriminatory Algorithmic Mechanism Design Based WWW Content Replication. Informatica 31(1), 105–119 (2007)
22. Khan, S.U., Ahmad, I.: Game Theoretical Solutions for Data Replication in Distributed Computing Systems. In: Rajasekaran, S., Reif, J. (eds.) Handbook of Parallel Computing: Models, Algorithms, and Applications, vol. ch. 45. Chapman & Hall/CRC Press, Boca Raton (2007) ISBN: 1-584-88623-4
23. Khan, S.U., Ahmad, I.: A Semi-Distributed Axiomatic Game Theoretical Mechanism for Replicating Data Objects in Large Distributed Computing Systems. In: 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS), Long Beach, CA, USA (March 2007)
24. Khan, S.U., Ahmad, I.: Replicating Data Objects in Large-scale Distributed Computing Systems using Extended Vickery Auction. International Journal of Computational Intelligence 3(1), 14–22 (2006)

25. Khan, S.U., Ahmad, I.: Data Replication in Large Distributed Computing Systems using Supergames. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), Las Vegas, NV, USA, pp. 38–44 (June 2006)
26. Khan, S.U., Ahmad, I.: A Pure Nash Equilibrium Guaranteeing Game Theoretical Replica Allocation Method for Reducing Web Access Time. In: 12th International Conference on Parallel and Distributed Systems (ICPADS), Minneapolis, MN, USA, pp. 169–176 (July 2006)
27. Khan, S.U., Ahmad, I.: A Powerful Direct Mechanism for Optimal WWW Content Replication. In: 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Denver, CO, USA (April 2005)
28. Khan, S.U., Ahmad, I.: Replicating Data Objects in Large Distributed Database Systems: An Axiomatic Game Theoretical Mechanism Design Approach. Distributed and Parallel Databases 28(2-3), 187–218 (2010)
29. Khan, S.U., Ahmad, I.: A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids. IEEE Transactions on Parallel and Distributed Systems 20(3), 346–360 (2009)
30. Khan, S.U., Maciejewski, A.A., Siegel, H.J., Ahmad, I.: A Game Theoretical Data Replication Technique for Mobile Ad Hoc Networks. In: 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS), Miami, FL, USA (April 2008)
31. Kołodziej, J., Xhafa, F., Kolanko, Ł.: Hierarchic Genetic Scheduler of Independent Jobs in Computational Grid Environment. In: Otamendi, J., Bargieła, A., Montes, J.L., Doncel Pedrera, L.M. (eds.) Proc. of 23rd ECMS, Madrid, pp. 108–115. IEEE Press, Dudweiler (2009)
32. Kołodziej, J., Xhafa, F.: A Game-Theoretic and Hybrid Genetic meta-heuristic Model for Security-Assured Scheduling of Independent Jobs in Computational Grids. In: Proc. of CISIS 2010, pp. 93–100. IEEE Press, USA (2010)
33. Kołodziej, J., Xhafa, F., Bogdański, M.: Secure and task abortion aware GA-based hybrid metaheuristics for grid scheduling. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 526–535. Springer, Heidelberg (2010)
34. Kołodziej, J., Xhafa, F.: Meeting Security and User Behaviour Requirements in Grid Scheduling. Simulation Modelling Practice and Theory 19(1), 213–226 (2011), doi:10.1016/j.simpat.2010.06.007
35. Kołodziej, J., Xhafa, F.: Integration of Task Abortion and Security Requirements in GA-based Meta-Heuristics for Independent Batch Grid Scheduling. Computers and Mathematics with Applications (2011), doi: 10.1016/j.camwa.2011.07.038
36. Kołodziej, J., Xhafa, F.: Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. Future Generation Computer Systems 27, 1035–1046 (2011), doi:10.1016/j.future.2011.04.011
37. Kołodziej, J., Khan, S.U., Xhafa, F.: Genetic Algorithms for Energy-aware Scheduling in Computational Grids. In: Proc. of the 6th IEEE International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing (3PGCIC), Barcelona, Spain (October 2011)
38. Kosar, T., Balman, M.: A new paradigm: Data-aware scheduling in grid computing. Future Gener. Comput. Syst. 25(4), 406–413 (2009)
39. Liu, H., Abraham, A., Xhafa, F.: Peer-to-Peer Neighbor Selection Using Single and Multi-objective Population-Based Meta-heuristics. In: Xhafa, F., Abraham, A. (eds.) Metaheuristics for Scheduling in Distributed Computing Environments. SCI, vol. 146, pp. 323–340. Springer, Heidelberg (2008)

40. Liu, H., Orban, D.: GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications. In: 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID), pp. 295–305 (2008)
41. Pinel, F., Pecero, J.E., Bouvry, P., Khan, S.U.: A Two-Phase Heuristic for the Scheduling of Independent Tasks on Computational Grids. In: Proc. of ACM/IEEE/IFIP International Conference on High Performance Computing and Simulation (HPCS), Istanbul, Turkey (July 2011)
42. Ranganathan, K., Foster, I.: Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In: Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC), Edinburgh, Scotland. IEEE CS Press, Los Alamitos (2002)
43. Shatdal, A., Kant, C., Naughton, J.F.: Cache conscious algorithms for relational query processing. In: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994), Santiago, Chile, pp. 510–521. Morgan Kaufmann Publishers, Inc., San Francisco (1994)
44. Valentini, G.L., Lassonde, W., Khan, S.U., Min-Allah, N., Madani, S.A., Li, J., Zhang, L., Wang, L., Ghani, N., Kołodziej, J., Li, H., Zomaya, A.Y., Xu, C.-Z., Balaji, P., Vishnu, A., Pinel, F., Pecero, J.P., Kliazovich, D., Bouvry, P.: An Overview of Energy Efficiency Techniques in Cluster Computing Systems. Cluster Computing (2011), doi:10.1007/s10586-011-0171-x
45. Venugopal, S., Buyya, R.: An SCP-based heuristic approach for scheduling distributed data-intensive applications on global grids. J. Parallel Distrib. Comput. 68, 471–487 (2008)
46. Venugopal, S., Buyya, R., Kotagiri, R.: A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing (2009)
47. Wang, L., Khan, S.U.: Review of Performance Metrics for Green Data Centers: A Taxonomy Study. Journal of Supercomputing, 1–18 (2011), doi:10.1007/s11227-011-0704-3
48. Wasson, G., Humprey, M.: Policy and enforcement in virtual organizations. In: Proceedings of the 4th International Workshop on Grid Computing, Phoenix, Arizona, IEEE CS Press, Los Alamitos (2003)
49. Xhafa, F., Abraham, A.: Computational models and heuristic methods for grid scheduling problems. Future Generation Computer Systems 26, 608–621 (2010)
50. Xhafa, F., Carretero, J., Barolli, L., Durresi, A.: Immediate Mode Scheduling in Grid Systems. International Journal of Web and Grid Services 3(2), 219–236 (2007)
51. Xhafa, F., Barolli, L., Durresi, A.: Batch Mode Schedulers for Grid Systems. International Journal of Web and Grid Services 3(1), 19–37 (2007)
52. Zhang, J., Lee, B., Tang, X., Yeo, C.: Impact of Parallel Download on Job Scheduling in Data Grid Environment. In: Proc. of the Seventh International Conference on Grid and Cooperative Computing, pp. 102–109 (2008)
53. Zeadally, S., Khan, S.U., Chilamkurti, N.: Energy-Efficient Networking: Past, Present, and Future. Journal of Supercomputing, 1–26 (2011), doi:10.1007/s11227-011-0632-2

Improving Scalability of an Hybrid Infrastructure for E-Science Applications

Olivier Terzo, Lorenzo Mossucca, Pietro Ruiu, Giuseppe Caragnano, Klodiana Goga, Riccardo Notarpietro, and Manuela Cucca

Infrastructure and Systems for Advanced Computing (IS4AC),
Istituto Superiore Mario Boella, Via Pier Carlo Boggio 61, 10138 Torino, Italy
Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy
{terzo,mossucca,ruiu,caragnano,goga}@ismb.it
{riccardo.notarpietro,manuela.cucca}@polito.it
http://www.ismb.it

Abstract. The Italian GPS receiver for Radio Occultation has been launched from the Satish Dhawan Space Center (Sriharikota, India) on board of the Indian Remote Sensing OCEANSAT-2 satellite. The Italian Space Agency has established a set of Italian universities and research centers to develop an innovative solution in order to quickly elaborate RO data and extract atmospherical profiles. The algorithms adopted can be used to characterize the temperature, pressure and humidity. In consideration of large amount of data to process, an hybrid infrastructure has been created using both the existing grid environment (fully physical) and the virtual environment composed of virtual machines from local cloud infrastructure and Amazon EC2. This enhancement of the project stems from the need of computational power in case of an unexpected burst of calculation that the physical infrastructure would not be able to respond on its own. The virtual environment implemented guarantees flexibility and a progressive scalability and other benefits derived by virtualization and cloud computing.

Keywords: radio occultation, grid computing, hybrid architecture, virtualization, scheduling.

1 Introduction

The Italian Space Agency (ASI) [1] developed a new GPS receiver devoted to Radio Occultation (RO). The space-based GPS limb sounding, conventionally known as GPS Radio Occultation, is a remote sensing technique for the profiling of atmospheric parameters (first of all refractivity, but also pressure, temperature, humidity and electron density, see [14,11]). It is based on the inversion of GPS signals collected by an ad hoc receiver placed on-board a Low Earth Orbit (LEO) platform, when the transmitter rises or sets beyond the Earths limb. The relative movement of both satellites allows a quasi vertical atmospheric scan of the signal trajectory and the profiles extracted are characterized by high vertical resolution and high accuracy. The RO technique is applied for meteorological purposes (data collected by one LEO receiver placed at 700 km altitude produce 300-400 profiles per day, worldwide distributed) since such observations can easily be assimilated into Numerical Weather Prediction models. Anyway, it is also very useful for climatological purposes, for gravity wave observations and for Space Weather applications. This will cause the phase of the signal to be delayed. Moreover, the bent Geometric Optics trajectories followed by the signal during an entire occultation event will span the entire atmosphere in the vertical direction. As a consequence, through the inversion of the phase delay measurements, the refractivity related to each trajectory perigee can be evaluated, and a vertical profile can be identified. From refractivity, and adopting variational techniques, temperature and water vapour profiles can also be inferred. Given the characteristics of global coverage, good accuracy and high vertical resolution, products derived using such a technique are operationally used in input to weather forecasting model tools, and could also be harnessed in monitoring climate changes. The Italian Space Agency funded a pool of Italian Universities and Research Centers for the implementation of the overall RO processing chain, which is called ROSA-ROSSA (ROSA-Research and Operational Satellite and Software Activities). The ROSA-ROSSA is integrated in the operational ROSA Ground Segment it is operating in Italy (at the ASI Space Geodesy Center, near Matera) and in India (at the Indian National Remote Sensing Agency [9], near Hyderabad) starting from the 2009 autumn season. The Italian GPS receiver for Radio Occultation (ROSA) files in the forthcoming OCEANSAT-2 Indian mission. In this framework, Italian Space Agency has funded the development of the operational RO Ground Segment, which include the ROSA-ROSSA software (ROSA-Research, Operational Satellite and Software Activities). Partners of this project are several Italian universities research groups, research centres and an industry, which are responsible for the development and the integration of the various software modules defining the ROSA-ROSSA software: Politecnico of Turin, CISAS (University of Padua), University "La Sapienza" of Rome, University of Camerino, International Center of Theoretical Physics (ICTP) of Trieste, Istituto dei Sistemi Complessi (ISC) of Florence, Innova (Consorzio per l'Informatica e la Telematica) of Matera. The ROSA Ground Segment will be implemented in two different ways. It is implemented in a distributed architecture, through a hybrid infrastructure called the Hybrid Processing Management (HPM).

The paper is structured as follows: Section 2 explains the related work and shows the architecture. Section 3 describes the project background: scientific context, virtualization and cloud environment. Section 4 presents hybrid architecture. Section 5 depicts some performance tests and the last section draws the conclusion and future work.

2 Related Work

The existent system is managed by an integrated software, called Grid Processing Management (GPM), devoted to handle and process data of the OCEANSAT-2

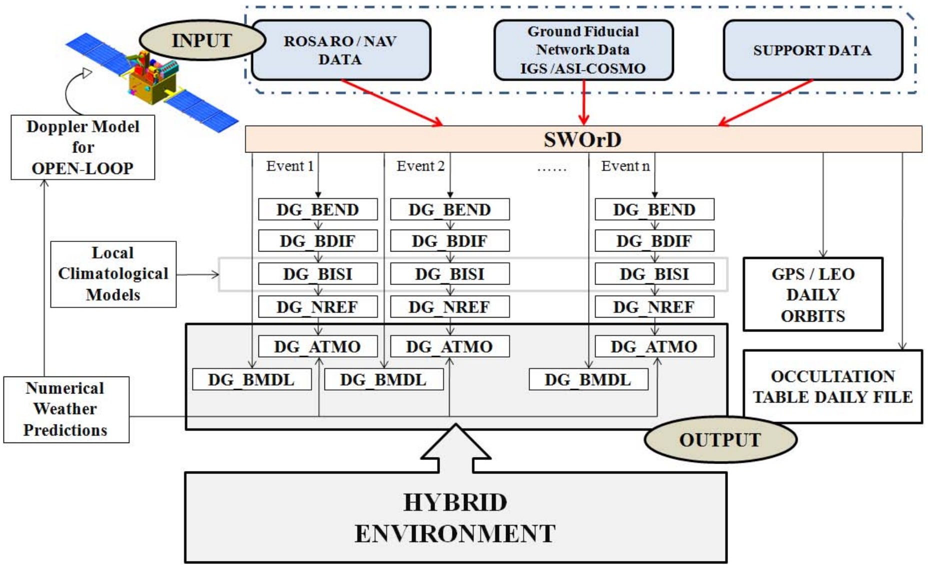

INPUT
ROSA RO / NAV DATA
Ground Fiducial Network Data IGS / ASI-COSMO
SUPPORT DATA
Doppler Model for OPEN-LOOP
SWOrD
Event 1
Event 2
......
Event n
DG_BEND
DG_BDIF
DG_BISI
DG_NREF
DG_ATMO
DG_BMDL
Local Climatological Models
GPS / LEO DAILY ORBITS
OCCULTATION TABLE DAILY FILE
Numerical Weather Predictions
OUTPUT
HYBRID ENVIRONMENT

Fig. 1. Radio Occultation Data Flow

on board sensor. This architecture consists of the following components: worker nodes, repository, relational database, scheduler, agents and applications. The observed data, once acquired by the receiving ground station, are processed to produce refractivity, temperature and humidity profiles. The Radio Occultation events data processing consist of seven main steps, named Data Generators (DGs). All DGs are executed in series, these are SWOrD, DG_BEND, DG_BDIF, DG_BISI, DG_NREF, DG_ATMO and DG_BMDL (see Figure 1). The input and output files cover a 24 hours time interval. The outputs are daily data composed of about 256 occultation events to be processed in sequential way. For further details see [16]. In this context, where one needs to elaborate an enormous amount of data, using a grid architecture, there is already a great saving of time. But in some cases the system fills up, when all worker nodes are elaborating data, increasing execution time. A solution to solve this problem can be to use a dynamically scalable system using virtual machines. The architecture proposed also consists of a virtualized environment, which add to the grid virtual nodes on demand, in order to increase the computational power and to solve temporary peak processing. Also it allows to create multiple virtual nodes on the same machine optimizing physical resource, reducing energy consumption and decreasing maintenance costs. Virtualized systems also help to improve infrastructure management, allowing the use of virtual node template to create virtual nodes in a short time, speeding up the integration of new nodes on the grid and, therefore, improving the reactivity and the scalability of the infrastructure.

3 Project Background

3.1 Scientific Context for Radio Occultation

The ROSA-ROSSA software implements state of the art RO algorithms which are subdivided into seven different software modules, called Data Generators(DG), these are executed in a sequential mode. Starting from ROSA engineered data (or raw data observed by other RO payloads made available to the scientific community) coming from the ROSA on-board OCEANSAT-2 platform observations, from the ground GPS network (i.e. International GNSS Service network) and from other support data, the ROSA-ROSSA is able to produce data at higher levels, using a data processing chain defined by the following DGs:

SWOrD is a software module that fully supports the orbit determination, orbit prediction, and which implements data generation activities connected with the ROSA sensor on-board OCEANSAT-2. Input data for SWOrD are ROSA GPS navigation and Radio Occultation observations, ground GPS network data and other support data. It generates the following output data:
  - Estimated rapid orbits and predicted orbits for the GPS constellation in Conventional Terrestrial Reference frame;
  - Estimated rapid orbits and predicted orbits for the OCEANSAT-2 platform in Conventional Terrestrial and Celestial Reference frame;
  - 50 Hz closed-loop and 100 Hz Open-Loop excess phases and signal amplitudes data for each single occultation event;
  - Tables showing estimated and predicted (up to 6 hours in advance) occultation.

DG_BMDL predicts a bending angle and impact parameter profile usable as input in the ROSA on-board software Excess Doppler prediction module for open-loop tracking. For each "predicted" occultation event, latitude and longitude of the geometrical tangent points (the nearest point of each trajectory to the Earth's surface, evaluated through predicted orbits) is used to compute bending angle and impact parameter profile from interpolated numerical weather prediction models (bending angle and impact parameter are geometrical parameter univocally identifying each trajectory followed by the RO signal). Inputs for DG_BMDL are predicted GPS and LEO orbits, respectively, and Predicted Occultation Tables, together with ECMWF world forecasts for the synoptic times valid for the future observed occultation event.

DG_BEND provides raw bending angle and impact parameter profiles α(a) computed on GPS occulted signals on both GPS frequencies L_1 and L_2, by using a Wave Optics approach below a certain altitude. Above that altitude threshold, standard Geometrical Optics algorithms are applied. Inputs for DG_BEND are data (L_1 and L_2 Excess Phases and related orbit data) and data (L_1 and L_2 signal amplitudes).

DG_BDIF provides (for each event) a bending angle and impact parameter profile, on which the ionospheric effects have been compensated for. This DG processes both L_1 and L_2 bending angle and impact parameters profiles given as input, in order to minimize the first order ionospheric dispersive effects. Outputs for DG_BDIF are bending angle and impact parameter iono-free profiles.

DG_BISI provides profiles of bending angle versus impact parameter optimized in the stratosphere above 40 km. In the ROSA-ROSSA, data coming from a Numerical Weather Prediction Model are used in place of climatological data for implementing the statistical optimization procedure necessary to reduce the high noise level left to the signal after ionospheric first order compensation applied by the previous DG_BDIF. DG_BISI processes bending angle and impact parameter profiles obtained from DG_BDIF Data.

DG_NREF provides (for each event) the refractivity profile and dry air temperature and pressure profiles. This DG is able to process iono-free and properly initialized bending angle and impact parameter profiles in order to compute the corresponding dry air "quasi" vertical atmospheric profiles.

DG_ATMO allows to evaluate the temperature and the water vapour profiles using forecasts or analysis obtained by numerical weather prediction. This DG receives on input from DG_NREF data files and produces on output data files, which contain the total temperature and total pressure profiles in terms of wet and dry components.

3.2 Virtualization Overview

Virtualization is a technology that allows running several concurrent operating system instances inside a single physical machine, reducing the hardware costs and improving the overall productivity by letting many more users work on it simultaneously. The hypervisor, the fundamental component of a virtualized system, provides infrastructure support exploiting lower-level hardware resources in order to create multiple independent Virtual Machines (VM), isolated from each other. This virtualized layer, called also Virtual Machine Monitor (VMM), sits on top of the hardware and below the operating system. The hypervisor can control (create, shutdown, suspend, etc.) each VM that is running on top of the host machine. Multiple instances of different operating systems may share the virtualized hardware resources. The hypervisor is so named because it is conceptually one level higher than a supervisory program. A supervisory program or supervisor - also called kernel - is usually part of an operating system, that controls the execution of other routines and regulates work scheduling, input/output operations, error actions, and similar functions and regulates the flow of work in a data processing system.

There are two main virtualization approaches: Full Virtualization and Para Virtualization [19]. Full Virtualization provides emulation of the entire underlying hardware (CPU, memory, storage, etc.) to the VMs in order to start and run the operating system. These guests have no knowledge about the host OS since they are not aware that the hardware they see is not real but emulated. This approach, however, is burdened with a heavy overhead that affects the

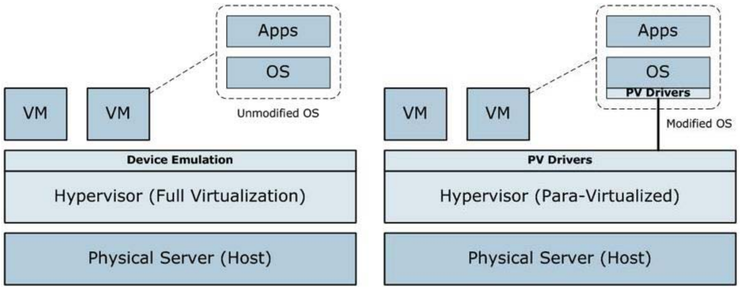


Fig. 2. Virtualization approach: Full Virtualization and Para Virtualization

system performances. The main advantage of this paradigm concerns the ability to run virtual machines with unmodified operating systems since the emulated hardware is completely transparent. A way to improve performances is the hardware-assisted virtualization (HVM) that enables efficient Full Virtualization using hardware capabilities (such as the Intel VT-x and AMD-V architectures) that provide direct platform-level architectural support for OS virtualization. For some hypervisors (like Xen and KVM) it is possible to recompile Para Virtualized drivers inside the guest machine running in HVM environment and load those drivers into the running kernel to achieve Para Virtualized I/O performance within an HVM guest. Para Virtualization approach uses a hypervisor for shared access to the underlying hardware but integrates virtualization aware code into the OS itself. In a context of Para Virtualization the guest operating system must be aware of being run in a virtual environment. So the original operating system, in particular its kernel, is modified to run in a Para Virtualized environment. The drawbacks of Full Virtualization are avoided by presenting a virtual machine abstraction that is similar but not identical to the underlying hardware. The main advantage is the execution speed, always faster than HVM and Full Virtualization approach.

Virtualization Benefits. As mentioned before virtualization allows to gain significant benefits from an economic and a resources' optimization point of view [21]. Besides these, other noteworthy benefits are:

– security, stability and isolation: it is possible to run services in a virtual environment totally independent from each other;
– environmental impact reduction: optimization of resources implies reduction of power consumption and cooling;
– administration and management simplification: due to the common virtualization layer and the adoption of snapshots (installation and configuration);
– disaster recovery: VM can be started up in few minutes and can be cloned and distributed in different locations;
– high reliability and load balancing improvement: thanks to snapshots and live migration features.

3.3 Cloud Environment

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [17]. The cloud is made up of four deployment models:

Private Cloud: The cloud infrastructure is operated exclusively for an organization. It can be a proprietary network or datacenter that uses cloud computing technologies such as virtualization. The private cloud can be managed by the organization or a third party and may exist on premise or off premise.

Public Cloud: The cloud infrastructure is made available in a pay-as-you-go manner to the general public or a large industry group and is owned by an organization selling cloud services (e.g., Amazon Web Services, Google AppEngine and Microsoft Azure).

Community Cloud: The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Hybrid Cloud: The cloud infrastructure is a composition of two or more cloud models (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Amazon Elastic Compute Cloud (EC2) is part of Amazon's cloud computing platform, Amazon Web Services (AWS). Amazon EC2 offers a web service through which users can boot an Amazon Machine Image (AMI) to create a virtual machine called "instance" and allows users to rent virtual machines. An AMI can be created from scratch by the user or can be used the pre-configured AMIs on Amazon AWS which can be modified and customized to suit the user's needs. An user can create, launch, and terminate instances as needed, paying by the hour for active instances, hence the term "elastic". Amazon EC2 is based on the XEN virtualization technology and sizes instances based on EC2 Compute Unit (ECU)[1] [18]. An EC2 instance may be launched with a choice of two types of storage for its root device. The first option, originally the only choice, is a local "instance-store" disk as a root device. The second option is to use an Elastic Block Storage (EBS) volume as a root device. By using Amazon EBS, data on the root device will persist independently from the lifetime of the instance. This enables users to stop and restart the instance at a subsequent time. Alternatively, the local instance store only persists during the life of the instance [18]. Elastic IP addresses are static IP (IPv4) addresses designed for dynamic cloud

[1] EC2 Compute Unit (ECU) - One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

computing. An Elastic IP address belongs to the account and not to a virtual machine instance. For the purposes of this project a Hybrid Architecture has been created which consists of a number of virtualized nodes (Private Cloud) and a number of Amazon EC2 node "instance" (Public Cloud), integrated in the exiting grid composed of physical nodes. In case of an unexpected computational power peak that the physical infrastructure and the virtualized nodes on the private cloud can't effort the global scheduler uses the AWS API to launch an EC2 instance based on an AMI containing all the necessary software (e.g., Globus Toolkit). For the project we have used an EBS Standard Large Instance (7.5 GB memory 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB instance storage, 64-bit platform).

4 Hybrid Architecture

The Hybrid Architecture (in Figure 3) consists of a number of virtualized nodes integrated into a grid composed of physical nodes. With physical node, we mean a machine that runs an operating system that has the exclusive use of the underlying hardware. The virtualized node is instead an instance of a virtual machine that can share resources with other nodes, managed by the hypervisor (see Figure 2). The project stems from need of computational power in case of an unexpected burst of calculation that the physical infrastructure would not be able to respond on its own. In these cases, where the physical grid has saturated its resources, the system asks to the hypervisor for new virtualized nodes which are allocated according to rules set in the scheduler that will be discussed later. This architecture allows to profit from the grid and from the virtualization (flexibility, scalability, cost reduction, etc.). The use of virtual nodes in addition to the physical nodes in the grid has considerable advantages. Virtual machines can be stopped and quickly reboot every time you need, without loss of information or problems to the chain flow of execution. Tests were performed on virtual machine to estimate startup time and the result is about 9500 ms. In addition to the reduction of startup time, the use of VM in the grid brings other benefits including load balancing and high availability. The load balancing allows migration of virtual machine from a physical box to another, in order to balance system performances; an high available system ensures migration of virtual machine when maintenance shall be paid on the server, avoiding possible (and usually lengthly) discontinuity in service provisioning. Startup time is the difference in milliseconds from the time when hypervisor receives a request to start the VM to the time when the VM is accessible on the network, and ready for a job execution.

4.1 Xen Overview

The Xen hypervisor is a layer of software that replaces the operating system running directly on the hardware of the computer. Xen [20] is an open-source VMM based on a Para Virtualization technique, released under GPL license for

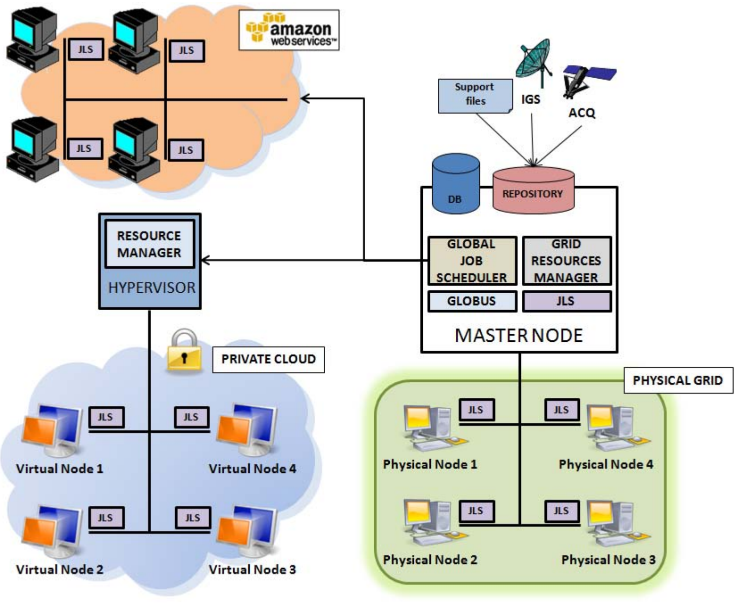


Fig. 3. Hybrid Architecture

x86- compatible machines, originally developed at the University of Cambridge. In this project it was decided to use Xen as virtualization platform because it is an open source product and is one of the few hypervisor that supports also Para Virtualization approach. As explained before the Para Virtualization technique uses a modified guest kernel that minimize virtualization overheads achieving higher performance than Full Virtualization approaches. The hypervisor, called domain0, is in charge of managing other guest VMs, called domainU: it is the direct interface between virtual machines and the hardware, and receives all requests for CPU, I/O and disk usage. Due to the separation between the OS and hardware, the hypervisor can run multiple independent operating systems safely and concurrently. The xm comands are the main interface for managing the guest domain. Thanks to them, it is possible to create, pause, and shutdown the VMs, but also enable or pin VCPUs and attach and detach virtual block devices. The commands are listened by a daemon called xend.

4.2 Resource Manager

Resource Manager (RM) is a component that optimizes computing resources needed to execute the job in the grid. It is a module that, once started, will automatically analyze the grid periodically; in this case each 5000 ms. It is designed and implemented in order to decide when allocate or deallocate new virtual machines as grid nodes. As mentioned above, in case of saturation of physical resources, the system is able to automatically startup virtual machines:

it can support the grid taking in charge the execution of jobs. This condition occurs when all active nodes of the grid are engaged in developing a process at least. This is not enough, in fact, saturation is reached when the sum of the processes on a machine (physical nodes of the grid) must be computationally heavy, and must occupy a lot of RAM. In other words, the Resource Manager defines a "resource profile" at regular time intervals getting information about CPU and RAM utilization of each virtual machine. The Resource Manager works on the system that hosts virtual machines and in our experiments it was not placed on the master node of the grid, but on a dedicated hardware box. It starts or stops (creates or destroys) a virtual machine using Xen API to communicate with the hypervisor. The calls are made through the XML-RPC protocol widely supported by the Xen project and also used by third-party tools such as libvirt. The RM allows to monitor and collect data which will be used for calculating system parameters and for the generation of log files. The logic model determines the allocation of the VMs, it is based on the observation of the status of the grid: specifically processes execution and machines availability. These information are retrieved by querying the database hosted on the master node of the grid. Periodically the nodes belonging to the grid, send information about its own status to the database located on the master node. These data are information about the state and the workload of the system (RAM allocated, average CPU usage). In particular, combining these information it is possible to obtain two fundamental elements: the status of the virtual machine and the parameter CUI (Computing Usage Index). The possible status of the virtual machine are two: the first is the status of *Available*, when a virtual machine is connected to the grid and has resources available to be able to perform the job (this means that the value of the CPU is between 0% and 5% use). The second status is *Running*, when a virtual machine has a queue of files to process c, where $c > 0$, and system resources are committed to process a job (the CPU is greater than 5% and above 70% for a long period of time). These data are used to calculate the CUI parameter that is an indication level about use of grid resources. It represents the ratio between the number of running nodes and the number of available nodes (Eq. 1).

$$CUI = \frac{\sum_{i=1}^{N} RunningNodes}{\sum_{i=1}^{N} AvailableNodes} \tag{1}$$

CUI is compared with bound values called start threshold and stop threshold. The first value, start threshold, indicates the saturation of the grid. If the CUI is greater than this value it is necessary to instantiate a new virtualized node. Resource manager starts a VM and sends to the master node information about availability of new node just started. As soon as the master node, where resides the job scheduler, detects the new virtualized node it can begin to assign the job. The stop threshold is the value that shows grid resources are underused and is therefore time to switch off the VM. Resource manager, once this limit is exceeded, tell to master node that the virtualized node is no longer available to receive job. Then it will proceed to shutdown the machine, after verifying that there are no file transfers in progress, there is not a queue of files to be processed, the processing job is actually completed.

4.3 Scheduling Model

The chain process is composed of 6 jobs developed by complex algorithms that involve a set of languages as Fortran, MatLab, C++, Mathematica, Java and Perl. After some tests with several schedulers like PBS and Condor, it was decided to develop in this infrastructure an ad-hoc job management scheduler: scheduling solutions tested didn't consider the system complexity due to several programming languages involved in the DGs, furthermore we focused on system scalability for hybrid infrastructure. A way to provide flexibility and scalability to the system is the implementation of a multi agents solution. The scheduler never gets information requiring the status to each nodes in fact on each node are installed two types of agents: Job and System agent. The first one is used to monitor the behavior of CPU, RAM and swap on nodes during the DG execution, and when DG is finished, it handles sending these information to master node. The second one, the System agent, is used to monitor the availability of each service on the node and periodically sends to the master node its status: if all services are available the node is ready to receive a job. The advantage is that the scheduler retrieves a pre list of available node (and ready for execution) only with a simple query to the database. The scheduling process is split in the following phases (see Figure 4):

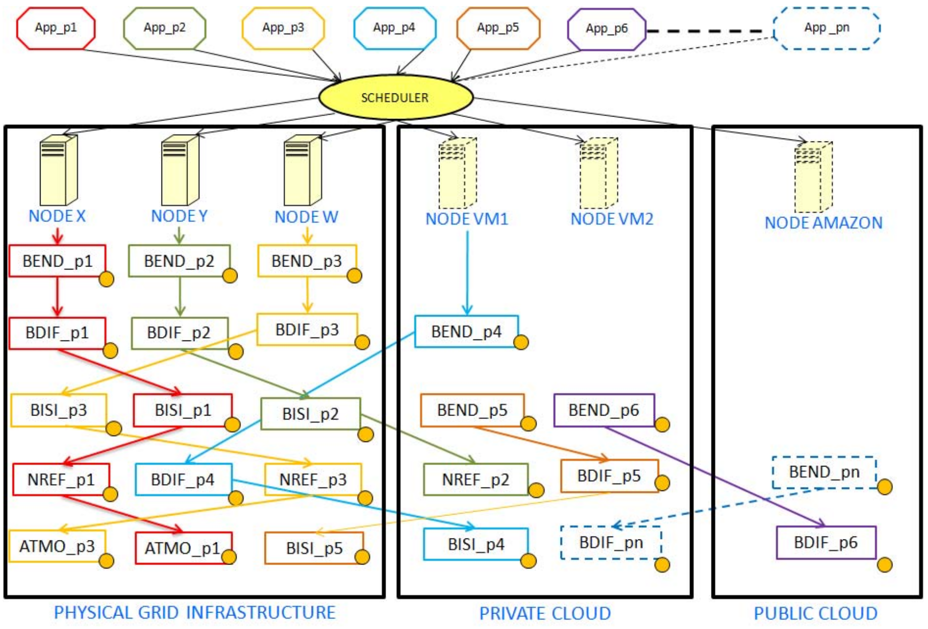


Fig. 4. Scheduling Approach

- resources discovery is directly related to the information sent by the agents in order to return a pre list of available nodes;
- in case of saturation of the physical grid, when no physical nodes are free for execution, the scheduler asks for new nodes to the virtualized infrastructure. A specific local resource scheduler daemon was developed for checking periodically request of new virtualized nodes;
- when there are not available nodes on the traditional grid computing environment and on virtualized infrastructure, the Resource Manager provides to add new virtual nodes from Amazon EC2.

This scheduling approach guarantees an automatic upscaling of the computational capacities. On the same way, an automatic downscaling mechanism was implemented: on virtualized infrastructure when a virtual grid node completed assigned jobs, the Resource Manager provides to shutdown the virtual nodes. On the same way in Amazon EC2, the Resource Manager provides to shutdown the instances.

5 Performance Test

During the test phase, we evaluated the elaboration time of each Data Generators executed on two types of nodes: physical and virtualized node. The server used is equipped with a dual-core Intel Xeon (4 CPU), 8 GB of RAM and 130 GB of storage. The operating system is Ubuntu server. The guest machines reside entirely on this server and therefore they share the resources (RAM, CPU, disk): each machine has 2 GB of RAM and 2 dedicated CPUs. Virtualized nodes are configured exactly like a physical node of the grid. It has been installed the softwares used for the chain processing and some system tools for the local job scheduling and monitoring of the resources. It was decided to use Para Virtualized systems since it was shown that (in terms of network and I/O), they have better performances than the Fully Virtualized one [10].

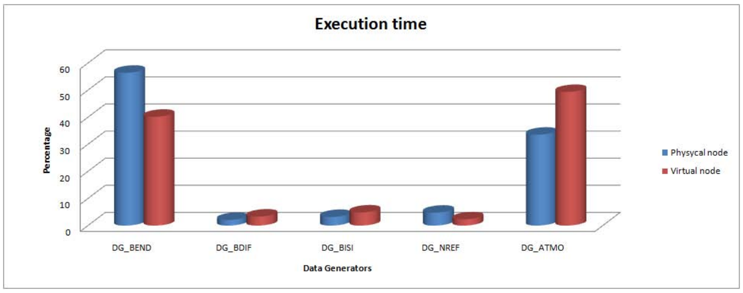


Fig. 5. Execution time

In Figure 5 a comparison of elaboration time is depicted. In Eq. 2 α is the weighted average elaboration time for each node belonging to the grid.

$$\alpha_i = \frac{\bar{t}_i}{\sum_{j=1}^{N} \bar{t}_j} \tag{2}$$

For each Data Generator, W represents a ratio between the virtual machine processing time and physical machine processing time (Eq. 3).

$$W_i = \alpha_i \frac{t\_virt_i}{t\_phy_i} \tag{3}$$

The execution time of algorithms DG_BDIF, DG_BISI, DG_NREF executed on the virtualized machine are almost comparable to the execution time on the physical machine. However, if the algorithm is executed on the virtualized machine DG_ATMO has a slight delay, estimated in 5%, compared with physical machines. The most important result is noticeable by observing DG_BEND: the execution time of this algorithm on the virtual machine is more quickly of about 23.5% than its execution on the physical machine (see Figure 6).

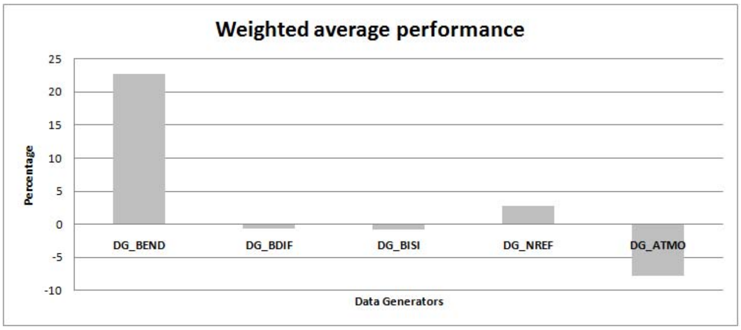


Fig. 6. Weighted average performance

Figure 7 depicts an estimation of the whole processing time calculated incrementing nodes number belonging to grid. When only one node is available, the total execution time for daily files is 1752 minutes (about 29 hours). Increasing the number of grid nodes the execution time decreases: note that just with two nodes the execution time is halved (912 minutes about 15 hour), but starting from 5 nodes the inclination of the curve is reduced and hence the gain time is lower. Although we have to consider that the grid introduces a time loss due to the file transfer time, needed to distribute the inputs to the worker nodes. Furthermore the virtualized environment brings a waste of time imputable to startup time of virtual node: as we saw above about 9500 ms for each VM.

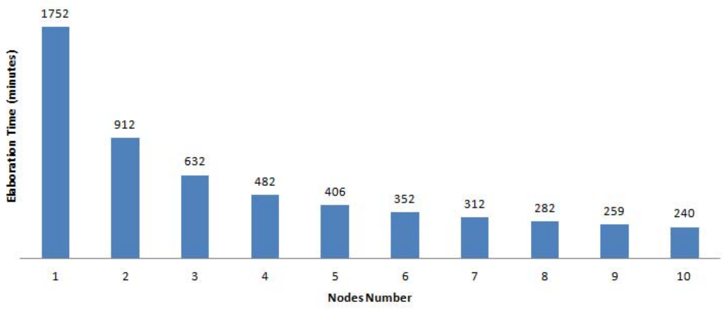


Fig. 7. Execution time in Hybrid Environment

6 Conclusion and Future Work

The ROSA-ROSSA software implements Radio Occultation technique, which runs for the first time on an hybrid infrastructure. This paper wants to be an improvement of a projects based on grid computing to solve temporary peak processing due saturation system. In frameworks such as Radio Occultation, where the amount of data to be processed is significant, the use of a hybrid architecture as the grid can be the best choice. We have focused on the implementation of a intelligent scheduler that can manage also the virtualized side of the infrastructure in order to assign jobs to nodes in an automatic way without any human interaction. When the algorithms are executed on virtual machines there is not a decline in system performance, but conversely, in the case of the algorithm DG_BEND, VMs have better performance than physical ones. Observing test results emerge that in a future scenario the Scheduler could assign the execution of DG_BEND algorithm only to virtual machines. By adopting this mechanism would be possible to reduce the execution times of the entire processing chain. Also, as future work we plan to extend the proposed architecture to computer clusters available across the European Grid Infrastructure (EGI).

Acknowledgment. The authors are grateful to the Italian Space Agency for supporting this project within the contract I/006/07/0 and to all ROSA-ROSSA partners for their contributions.

References

1. Italian Space Agency(ASI) (2011), `http://www.asi.it/`
2. Berman, F., Fox, G., Hey, A.: Grid computing making the global infrastructure a reality, pp. 117–170. Wiley, Chichester (2003)
3. Buyya, R., Abramson, D., Giddy, J.: NIMROD/G: An architeture of a resource management and scheduling system in a global computational grid. In: High Performance Computing Asia 2000, Beijing, China, pp. 283–289 (2000)

4. Dimitriadou, S., Karatza, H.: Job scheduling in a distributed system using backfilling with inaccurate runtime computation. In: International Conference on Complex, Intelligent and Software Intensive System, Washington DC, USA, pp. 329–336 (2010)
5. Foster, I., Kesselman, C.: The Grid 2: Blueprint for a New Computing Infrastructure, pp. 38–63. Morgan Kaufmann, San Francisco (2003)
6. The Globus Alliance (2011), `http://www.globus.org/`
7. The Globus Consortium (2011), `http://www.globusconsortium.org/`
8. Gradwell, P.: Grid scheduling with agents. In: Proceedings of the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2003), Melbourne, Australia, pp. 229–245 (2003)
9. Indian Space Research Organization (ISRO) (2011), `http://www.isro.org/`
10. Kurowski, K., Nabrzyski, J.A., Oleksiak, A., Weglarz, J.: Scheduling jobs on the grid multicriteria approach. Computational Methods in Science and Technology 12(2), 123–138 (2006)
11. Kursinski, E.R., Hajj, G.A., Schofield, J.T., Linfield, R.P., Hardy, K.R.: Observing Earth's atmosphere with radio occultation measurements using the Global Positioning System. Journal of Geophysical Research 102(D19), 23.429–23.465 (1997)
12. Leonid, O., Rupak, B., Hongzhang, S., Warren, S.: Job scheduling in a heterogeneous grid environment. Lawrence Berkeley National Laboratory (2004), `http://www.escholarship.org/uc/item/6659c4xj`
13. Luntama, J.P., Kirchengast, G., Borsche, M., Foelsche, U., Steiner, A., Healy, S., von Engeln, A., O'Clerigh, E., Marquardt, C.: Prospects of the EPS GRAS mission for operational atmospheric applications. Bulletin of the American Meteorological Society 89(12), 1863 (2008)
14. Melbourne, W.G., Davis, E.S., Duncan, C.B., Hajj, G.A., Hardy, K.R., Kursinski, E.R., Meehan, T.K., Young, L.E., Yunck, T.P.: The application of spaceborne GPS to atmospheric limb sounding and global change monitoring, pp. 18–94. JPL Publication (1994)
15. Wickert, J., Schmidt, T., Beyerle, G., Knig, R., Reigber, C., Jakowski, N.: The radio occultation experiment aboard CHAMP: Operational data analysis and validation of vertical atmospheric profiles. Journal of the Meteorological Society of Japan 82(1B), 381–395 (2004)
16. Mossucca, L., Terzo, O., Molinaro, M., Perona, G., Cucca, M., Notarpietro, R.: Preliminary results for atmospheric remote sensing data processing through Grid Computing. In: The 2010 International Conference on High Performance Computing and Simulation (2010)
17. Mell, P., Grance, T.: The NIST Definition of Cloud Computing (Draft), `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`
18. Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/`
19. Bagliocco, S., Feruglio, D., Ramunno, G.: La virtualizzazione e i suoi aspetti di sicurezza. Assosecurity, 18–20 (2010)
20. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization. University of Cambridge Computer Laboratory (2003)
21. The Future Of Cloud Computing, Expert Group Report, European Commmission, Information Society and Media, pp. 14–15 (2010)

Energy Aware Communication Protocols for Wireless Sensor Networks

Ewa Niewiadomska-Szynkiewicz[1,2]

[1] Institute of Control and Computation Engineering,
Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
[2] Research and Academic Computer Network (NASK),
Wawozowa 18, 02-796 Warsaw, Poland
ens@ia.pw.edu.pl, ewan@nask.pl

Abstract. The ad hoc networking is an ultimate technology in wireless communication that allows wireless devices located within their transmission range to communicate directly to each other without the need for established fixed network infrastructure. It is a new area of research that has become extremely popular over the last decade and is rapidly increasing its advance into different areas of technology. In this paper[1] properties, limitations and basic issues related to development of wireless sensor network applications are investigated. The focus is on reliable and energy aware inter-node communication strategies. The approaches to power control and activity control of nodes are briefly summarized. The results of the performance evaluation of energy aware protocols through simulation are presented and discussed. The protocol that relies on hierarchical routing and uses a periodic coordination for energy efficient WSN is described and investigated.

Keywords: Ad hoc network, wireless sensor network, energy aware communication, energy balancing, topology control.

1 Introduction to Ad Hoc Network

An ad hoc network is a wireless decentralized structure network comprised of nodes, which autonomously set up a network. No external network infrastructure is necessary to transmit data – there is no central administration. Freely located network nodes participate in transmission (connected devices can act as end terminal or else an intermediate transmission point – the router). The network nodes can travel in space as time passes, while direct communication between each pair of nodes is usually not possible. Generally, an ad hoc network can consist of different types of devices.

Ad hoc networks can be classified by their application. Two common types of ad hoc network are: Wireless Sensor Network (WSN) and Mobile Ad hoc Network (MANET). WSN is a distributed system composed of small-size, embedded

[1] This work was partially supported by National Science Centre grant NN514 672940.

devices grouped into network nodes deployed densely over a significant area. In most cases WSNs are stationary or quasi-stationary, while node mobility can be ignored. MANET comprises self-configuring mobile wireless communication devices which combine the roles of terminals and routers. Each device can run applications and participate in transferring data to recipients within its range. Network nodes can move in space. The lack of fixed network infrastructure components both in WSN and MANET allows creating unique topologies and enables the dynamic adjustment of individual nodes to the current network structure in order to execute assigned tasks. However, for the protocols to operate in this mode in practice, several basic issues must be solved. The most important ones are:

– Limited resources. Nodes comprised by the network are often small battery-fed devices, which means their power source is limited. The network's throughput is also limited.
– Poor quality of connection. The quality of wireless transmission depends on numerous external factors, like weather conditions or landform features. Part of those factors change with time.
– Unstructured and time-varying network topology. Each network node can at any time, for whatever reason, leave the network. Similarly, the node can rejoin the network at any time during its operation.
– Problems with ensuring secure network operation. The spontaneous nature of ad hoc network makes it vulnerable to any external threats and attacks. Using a computer with a wireless network adapter, any outsider can gain access to an unprotected network. Maintaining an appropriate level of security requires solving many issues that are not present in traditional computer networks.

The other constraints are limited computation power and memory of a single node, limitations in sensor accuracy, operation in hostile environments. Therefore, design and development of ad hoc networks is a non-trivial task. The main directions of current research in ad hoc networking include increasing the potential of hardware components in terms of smaller size of devices and their cost, accurate location systems for calculating positions of devices [2,4,25,26], energy-aware communication [2,3,23,29], dedicated operating systems and simulation software [19,21,31], efficient protocols and algorithms [17,18,27,29,33].

The main contribution of this paper is to point out the problems concerned with energy-aware communication in WSN, a topic that has been a subject of intensive research in recent years. In section 2, we provide the introduction to WSN applications, i.e., clustering WSNs, communication standards and topology control. Next, we briefly summarize various approaches to energy aware communication, i.e., technologies and protocols. In section 3 we describe and evaluate protocols based on power control and activity control techniques, and discuss the proposed modification to improve the performance of the well known routing technique.

2 Wireless Sensor Networks

In literature one can find an extensive survey and classification of the state of the art in WSN hardware and software. Depending on the application, WSN must support quality of service aspects such as real time constraints, maximal lifetime, dependable communication, robustness, tamper resistance, eavesdropping resistance, etc. All listed requirements impact on the key dimensions of the design: optimal sensor deployment, accurate localization of sensors, reliable and energy aware data transmission. In the case of networks comprising several hundreds or thousands of nodes, it is necessary to choose an architecture and technology, which will enable relatively cheap production of individual devices. For this reason, the capacities of sensors are seriously limited. Hence, small size batteries are the most frequent power source. Individual WSN network nodes can collect data recorded by sensors but usually do not have enough power to process it. Moreover, analysis require collection of information from many points. Therefore, efficient and energy aware inter-node communication is necessary in order to transfer data to the base station.

2.1 Clustering WSNs

Grouping the nodes into disjoint and mostly non-overlapping clusters is an effective approach to manage large scale WSN. It supports network scalability and improves stability and efficiency of performance. Many clustering schemes are provided. They can be classified with respect to various criteria, e.g., network architecture, network operation or objective of the node grouping process and cluster head selection. They can be compared based on various metrics such as cluster stability and scalability, cluster overlapping, load balancing, location awareness, support for node mobility. A taxonomy, general classification and a survey of clustering schemes is provided in [1]. Generally, a cluster formation in WSN is based on the following characteristics: every node has to be connected to some clusters, nodes in a cluster must be able to communicate with others, often maximum diameter of all clusters in the network is the same. Most algorithms form clusters in distributed way through local broadcasts with a maximum one or several (not many) hops. The cluster size is adapted to network capability and objectives. The cluster head is usually pre-assigned or picked randomly from the deployed set of nodes. Clustering algorithms are used in order to facilitate meeting application requirements. The popular objectives for WSN clustering are: load balancing, fault tolerance, distributed localization systems, increased connectivity and maximal lifetime of the network. Hence, many energy aware routing protocols assume grouping sensor nodes into clusters.

2.2 Communication Methods

Small communication range and poor quality of connection in WSNs result in communication limitations. Each node communicates only with the nodes present in its closest vicinity (the neighbors). For this reason, the natural communication method in wireless sensor networks is the multi-hop routing. When using the multi-hop routing, it is assumed that the receiving node is not located within the transmitter's range. Contrary to single-hop networks, the transmitter must transmit data to the receiver by means of intermediate nodes. This is a certain limitation that hinders the implementation of routing algorithms but enables the construction of network of greater capacity. A Multi-hop network enables simultaneous transmission via many independent routes. The independence of routes reduces the interference between individual nodes, which additionally enhances the wireless transmission speed in comparison to single-hop networks, where devices share common space.

Communication protocols used in modern wireless networks like IEEE 802.11, IEEE 802.15.1 (Bluetooth) or IEEE 802.15.4 (ZigBee) enable the ad hoc mode operation. Medium Access Control (MAC) protocols guarantee efficient access to the communication media while carefully managing the energy allotted to the node. This goal is typically achieved by switching the radio to a low-power mode based on the current transmission schedule. The comprehensive summary of MAC protocols for WSNs is presented in [3,17,30]. The results of simulations that show their capabilities and efficiency in terms of the energy consumption are described in [30].

2.3 Topology Control

The locations of nodes that are available for communication define a WSN topology. In general, the objective of topology control techniques is to generate network topologies with desired properties (e.g. connectivity, coverage, lifetime, etc.) while reducing node energy consumption and/or increasing WSN capacity [2,24,29]. It should be pointed out that the term *topology control* has been used with different scope by different authors. For Anastasi et al. [3] the basic idea behind topology control is to exploit the network redundancy, i.e., topology control techniques calculate the optimal set of nodes that guarantee the network connectivity. For Santi [29] topology control relies on power control. He presents an overview of techniques, in which nodes setting their transmit power level make local decisions regarding their transmission range. Some authors include in topology control clustering techniques. According to Akyildiz and Vuran [2], in WSNs a topology refers to locations, power and activity states of nodes. They list four main approaches for topology control: *deployment* – determining the positions of nodes in a network in an efficient way, *clustering* – grouping sensor nodes into clusters to improve energy efficiency, *activity control* – activation and deactivation of nodes, *power control* – setting the transmit power of a transceiver to maintain a transmission range. In general, topology control is a crucial part of energy aware communication protocols applied to WSNs.

3 Energy-Aware Communication Protocols

To maximize a lifetime of WSN, all aspects such as: architecture, circuits, protocols and algorithms must be made energy efficient. In this paper we focus on energy conservation schemes and energy aware communication protocols. They differ in energy efficiency objectives, e.g.: minimizing an energy consumption in forwarding an individual packet, minimizing the total energy consumption in the whole network and balancing power consumption in a network. All these protocols implement various schemes and optimization techniques, they utilize clustering solutions, topology control and different routing schemes. Different authors propose different classifications and taxonomies of energy conservation schemes [2,3,17,29]. Several protocols that utilize clustering schemes, power and activity control of network nodes are described in the following subsections.

3.1 Power Control Protocols

The common approach to energy efficient communication is to control a transmit power of each node to maintain the communication range of a node. By controlling the transmit power we can influence a network lifetime, connectivity, interference and latency. The power required for two network nodes x_i and x_j to exchange a message is proportional to d_{ij}^2, where d_{ij} denotes the distance between them. Lets assume that instead of performing direct transmission, a relay node x_r is used. In such case two transmissions need to be performed: from a source node x_i to a relay node x_r (distance d_{ir}) and from the node x_r to the destination node x_j (distance d_{rj}). Let us consider a triangle $x_i x_r x_j$, also let α be an angle at vertex x_r. By elementary geometry we have:

$$d_{ij}^2 = {d_{ir}}^2 + {d_{rj}}^2 - 2 d_{ir} d_{rj} \cos\alpha \tag{1}$$

When $\cos\alpha \leq 0$, total amount of energy spent to transmit a data package is smaller when a relay node is used. Hence, short transmissions in the network are desired. They will involve smaller power consumption and have smaller impact on other, simultaneously effected, transmissions, thus increasing the network throughput. Power control techniques [2,29] assume that the nodes have impact on the power used to transmit a message. The basic task consists in attributing the level of power used to send messages to every node in order to minimize the amount of power received from the power source, while at the same time maintaining the coherence of the network. Power control protocols (PC) are responsible for providing the routing protocols with the list of nodes' neighbors, and making decisions about the ranges of transmission power utilized in each transmission. Therefore, the power control layer is placed partially in the *network layer* and the *data link layer* in the OSI model.

PC protocols may utilize various information about a network and nodes – their neighbors and resources. We can distinguish various types of such protocols. They can be classified into several groups, the most popular are listed in Table 1. Location-based protocols build a topology based on information about

Table 1. Power control protocols

Location-based	Direction-based	Neighbor-based
R&M [28]	CBTC [34]	KNEIGH [7], XTC [35]
LMST [22]	DistRNG [8]	LMA [20], LMN [20]

the geographical location of all nodes, which is assumed to be available to each node in a network. Direction-based protocols utilize less accurate data – they relay on the ability of all nodes to estimate the relative direction of their neighbors. Neighbor-based techniques determine all neighbors within the maximum transmitting range and build an order on this set of neighbors. Hence, they rely on the ability of all nodes to determine and identify their neighbors. It is obvious that all PC protocols form various communication topologies. To examine this we implemented and tested two location-based protocols, i.e., LMST (*Local Minimum Spanning Tree*) developed by Li, Wang and Song, introduced in [22], and R&M described by Rodoplu and Meng in [28]. LMST and R&M protocols implement different algorithms, and are dedicated to different application scenarios. LMST is composed of three phases: information exchange, topology creation and transmission. Each node sends a broadcast message, at maximum transmit power, containing its ID and location information to its one hop neighbor. These messages are periodically broadcasted during protocol execution. As a result, each node determines a set of its neighbors, and can calculate their locations. The nodes use neighborhood information to generate their local Minimum Spanning Trees (MSTs). Finally, each node determines the transmit power to its neighbors in its MST. R&M builds a topology that is optimized for the all-to-one communication pattern. It calculates the most energy efficient path from any nodes to the master one in two phases. In the first phase each node in the network computes the set of its neighbors and enclosure[2]. In the second phase the minimum-energy reverse spanning tree rooted at the master node is calculated.

Figure 1 shows the suggested paths for data transmitting calculated by LMST and R&M. They can be compared with the topology generated without utilizing any power control algorithm (Fig. 2). We can observe that the LMST and R&M protocols select the paths minimizing the energy consumption from all possible paths (w.r.t. radio range connections) presented in Fig. 2. The topology formed by LMST provides alternative paths, see Fig 1. We performed a variety of simulation experiments to cover a wide range of network system configurations including: size of the network, nodes deployment and density. The key metric for evaluating LMST and R&M was the energy consumption used for data transmission. All experiments were conducted using the ns-2 simulator (www.is.edu/nsnam/ns/). The selected results of simulations for WSN formed by 300 nodes (simulating the MICAz mote manufactured by Crossbow) with

[2] The region beyond which it is not energy efficient for the node to search for one-hop neighbors.

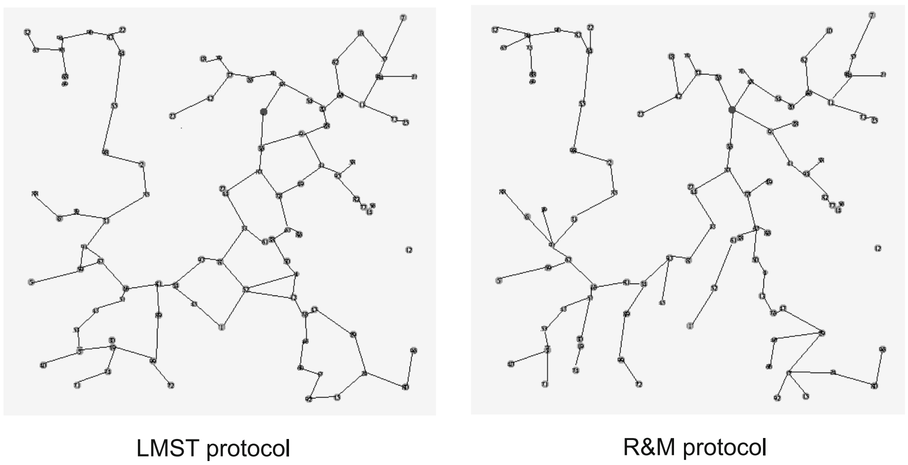


Fig. 1. Topology calculated using LMST and R&M protocols

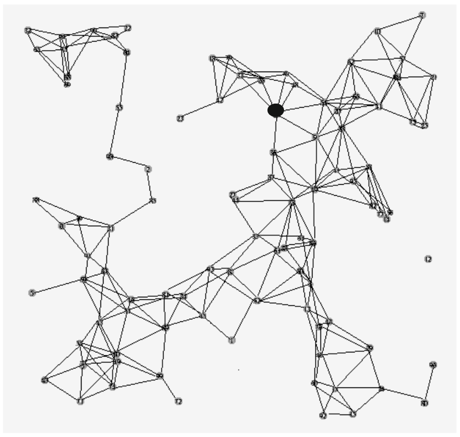


Fig. 2. Topology calculated without PC protocols

randomly generated positions in a square regions 400 × 400 to 3000 × 3000 are presented in Fig. 3. The initial energy resource of each node was equal to 21 kJ. The goal of each node was to send a single message that had to be delivered to the base station.

Figure 3 shows the average energy used by one node in WSN for data transmission in case of topologies built by R&M, and two variants of LMST: LMST0 (topology can contain unidirectional links), LMST1 (topology contains only bidirectional links). As a final observation we can say that R&M and LMST protocols can be successfully used to calculate optimal topology in many WSN application scenarios. Although both methods have to spend some energy to build the

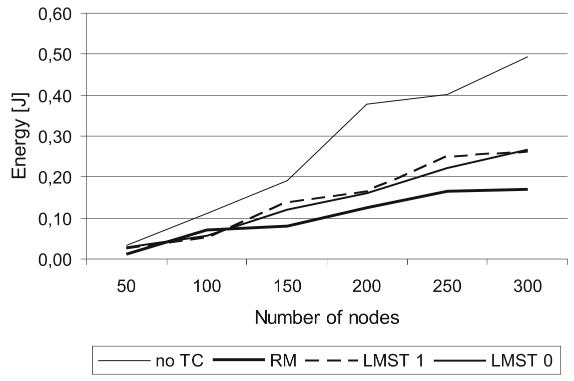


Fig. 3. Average energy consumption by one node for single transmission to the base station; different TC methods and network size

topology, which is concerned with exchanging of global information, which induces message overhead, they generate energy efficient topologies. The energy consumption for data transmission in case of a small size network (less than 120 nodes) is similar, while using topologies formed by R&M and LMST. In case of large size networks the R&M protocol seems to be much more efficient. In summary, both techniques generate different topologies and have some advantages and drawbacks. The R&M protocol can be successfully used in WSNs in which deployed sensors send messages to the master node (base station). The protocol relies on an explicit radio signal propagation model that is its potential disadvantage – the computed topology might be different from the optimal one if the actual channel conditions are different from those assumed by the channel model. LMST produces topologies with a smaller average node degree and average transmission radius with respect to those generated by R&M. The drawback of this protocol is that it can be used for networks, in which all nodes have the same maximum transmit power. In general, data transmission while applying the LMST protocol is more energy intensive, but formed topology is more robust and preserves connectivity in the worst case. Moreover, it can be computed in a fully distributed fashion.

3.2 Activity Control Protocols

Radio receiver in WSN network node can operate in one of four modes, which differ in the consumption of power necessary for proper operation: transmit – signal is transmitted to other nodes, receive – message from other node is received, idle – receiver inactive, turned on and ready to change to data transmission or receiving (low power consumption), sleep – radio receiver off. In literature and manuals one can find nominal power consumption and transmission range for available wireless cards. For example for popular MICAz transceiver the power consumption for various modes is as follows: transmit - 17.4 mA, receive - 19.6 mA, idle - 20 μA, sleep - 1 μA (http://www.hoskin.qc.ca/).

In order to extend the working time of an individual device (network node), it is frequent practice that some elements of this device are deactivated, including the radio receiver. They remain inactive for most time and are activated only to transmit or receive messages from other nodes. Hence, implementing the right policy for transition to various states is critical for effective idle power management. In general, formulating an optimum shutdown policy is a nontrivial problem.

The activity control (also known as power saving) protocols for WSNs employ dynamic management of radio device of all network nodes – in calculated intervals they put the radio receiver into the sleep mode. The objective is to limit the energy consumption while simultaneously minimizing the negative impact on the network throughput and on the efficiency of data transmission routing. These protocols should be capable of buffering traffic destined to the sleeping nodes and forwarding data in partial network defined by the covering set. The covering set membership needs to be rotated between all nodes in order to maximize the lifetime of the network. Various types of protocols and their implementations are used depending on the application scenarios, and are described in literature.

Connectivity-Based Protocols. Span [12] is a common distributed, connectivity-based protocol were nodes are partitioned into two groups: *coordinators* that stay awake continuously and forward messages, and *sleeping* nodes. The sleeping nodes periodically check their mode if needed to wake up and become a coordinator. The coordinators are adaptively selected from all nodes in a network using the eligibility rule that guarantees the sufficient number of coordinators in the area. In general, a node with a higher residual energy should be more likely to become a coordinator, coordinators should be elected in such a way to minimize their number. Span depends on the routing protocol. The routing protocol provides neighbors and connectivity data that are used to select coordinators.

Location-Based Protocols. The Geographic Adaptive Fidelity (GAF) protocol described in [36] assumes covering the network deployment area with virtual grid. The location information is employed to form clusters. GAF assumes that nodes in the particular grid cell are equivalent with respect to forwarding messages, so, any node within a cluster is able to relay a message to any node in any neighboring cluster. Hence, we can size the grid cell based on the nominal radio range r. This size is equal to $r/\sqrt{5}$. The concept of GAF is to maintain only one node with its radio transceiver turn on per grid cell. The function of an active state is rotated between all nodes in each cell. Whenever a node turns on its radio transceiver, it broadcasts a message containing its identifier and a value of its ranking function. The ranking function is assigned to each node. Its value depends on the energy left in of a given node. If a node receives a message from another node from the same cell with the higher value of the ranking function, it is allowed to turn its radio transceiver off.

In the Geographic Random Forwarding (GeRaF) protocol [10] nodes are periodically waked up to an active state, starting with a listening time. Hence, all nodes can forward messages if needed. The protocol operates in two phases: receiver-initiated forwarding phase and transmission phase. Whenever a node obtains a message to send it switches to the active state and broadcasts the packet containing its own location and the location of the receiver of the message. Each active node has a priority, which depends on the closeness to the destination (nodes located close to the destination have higher priority). A distributed randomization is used to reduce the situation, in which too many neighbors of all nodes simultaneously switch to a sleeping mode. Moreover, the part of the coverage area between the sender and the destination is divided into number of regions with associated priorities that depend on the closeness to the destination, too. Finally, the regions and nodes with the higher priorities are chosen to forward the message.

Clustering-Based Protocols. Many energy aware routing protocols rely on dividing a network into clusters with local cluster heads responsible for data aggregating and transmitting to the base station [1,5,11,14,15,32]. One of a common clustering-based approach is a Low-Energy Adaptive Clustering Hierarchy (LEACH) [16] – a self-organizing, adaptive protocol. It utilizes randomized rotation of cluster heads to evenly distribute energy load among all nodes in a network. Moreover, it performs local compression of data in each cluster to reduce global communication. In each cluster nodes elect themselves to be cluster head at a given time with a certain probability. LEACH is composed of four phases: cluster heads election, clusters formation, a TDMA (Time Division Multiple Access) schedule for all nodes in a cluster creation, data aggregation and transmission. The cluster heads are selected based on the amount of energy left in given nodes. The TDMA schedule allows the radio components of all non-cluster heads to be turned off at all times except during their transmit time. The main drawback of LEACH is that the random rotation of the cluster heads does not ensure their even distribution in a deployment area – hence multiple cluster heads can be concentrated in a small region. The modified version of cluster head selection algorithm used in LEACH is proposed in [32]. In this approach the probability of a given node to become a cluster head is based on its remaining energy level. The hybrid Energy-Efficient Distributed Clustering (HEED) protocol [37] considers explicitly energy resources while selecting cluster heads, too. The probability that the i-th node will be selected to become a cluster head is equal $P_i = N_{ch} E_i / E_i^{max}$, where N_{ch} denotes assumed percentage of cluster heads among all nodes, E_i current energy in the node i, E_i^{max} its maximum energy. L. Buttyan and P. Schaffer propose in [9] the Position-based Aggregator Node Election (PANEL) that brings one more step toward energy efficient clustering protocols. It assumes that the sensing field is divided into geographical clusters, and each node in a cluster is aware of the cluster location. The protocol was evaluated through extensive simulations. The authors claim that PANEL is more energy efficient and creates more cohesive clusters than HEED.

Hybrid Routing Algorithm with Special Parameters in Wireless Sensor Network (PRWSN) [13] is another clustering-based protocol. The created cluster formation has to satisfy special conditions with regard of the neighborhood principle and local information of each node and its neighbors. The cluster head is selected based on the scale of average local energy and the density of nodes surrounding a given node.

Hierarchical Protocols. It is common to apply the hierarchy to routing protocols. A hierarchical version of LEACH is the Energy Efficient Hierarchical Clustering (EEHC) described in [6]. In this approach data transmission to the destination is performed using multi-hop routing through multi-level hierarchy of clusters. We have developed a hierarchical version of GAF – a CGPS protocol that is described in the following section.

3.3 CGPS – Hierarchical Power Save Protocol

The CGPS (Coordinated Geographical Power Save) protocol relies on the GAF (Geographic Adaptive Fidelity) protocol [36] and a hierarchical routing. Similarly to GAF we assume covering the network deployment area with a virtual grid with the size of each cell equal to $r/\sqrt{5}$, where r denotes the radio range. The scheme of radio device management is taken from the GAF protocol, too. The novel idea is to decrease the energy usage not only deactivating the selected nodes but moreover putting selected grid cells to sleep and waking them up only when necessary. Hence, we introduce the coordination to power management in WSN, and distinguish two levels of activity control:

grid level - provides an algorithm of power management in each single grid cell (activation and deactivation of nodes in a given cell),

coordinator level - provides an algorithm of power management in the whole network (activation and deactivation of grid cells in WSN).

We propose to utilize the base station as a network coordinator. We assume that not every grid cell needs to maintain an active node. The main task of the coordinator is to select the cells in the grid that can be deactivated and estimate the optimal time period of their shutdown. Next, the coordinator calculates the possible paths for data transmission from all sensors to the base station taking into account only active grid cells according to the following algorithm.

The Routing Algorithm: The coordinator views the network grids as a graph. Each grid cell corresponds to a vertex in the graph and the connections to neighboring grid cells correspond to its edges. The nodes periodically send control messages to the coordinator. These messages contain the information about current state of their batteries. The coordinator assigns weights to the edges in the graph taking into account the current energy resources of all nodes. Let us assume that grids i and j are neighbors, let $pow(i)$ be the amount of energy left in the grid i. The weight assigned to the edge between grids i and j

is $min(pow(i), pow(j))$. Next, the minimum spanning tree on the graph with the coordinator as a root of the tree is calculated. The leaves of the tree are network grids that do not need to maintain an active node. The structure of a spanning tree was chosen in order to preserve the original network connectivity. The coordinator broadcasts the message that contains the current (computed) network topology (topology map) and the sequence number of the latest transmitted topology map, whenever newly calculated map differs from the previous one. In order to minimize the size of broadcasted messages transmitted by grid cells, the topology map is saved as a bitmap – one grid square is described by one bit, and dedicated broadcast algorithm was employed.

The *grid level* is responsible for implementing the policy for sleep-state transition of nodes in a single cell. The state transition algorithm provided in GAF is employed.

CGPS was evaluated via simulation using ns-2 software platform. It was compared with the plain GAF protocol and a network with no power save capabilities at all. The traffic scheme utilized during the experiments assumed random selecting of nodes sending messages to the base station at random time stamps. Figures 4 and 5 show the average amount of energy available per network node during the course of simulation in two experiments: networks with 60 (Fig. 4) and 90 nodes (Fig. 5) distributed uniformly over $800m \times 800m$ region, each node with initial energy resource 21 kJ. The simulations show that CGPS outperforms the GAF protocol in case of networks with small density of nodes. The amount of energy saved is greater than in the GAF protocol due to larger number of nodes with nonactive radio. When the density of network nodes increases, the amount of energy saved by CGPS falls to the level obtained for GAF (see Fig. 5). It is obvious that CGPS introduces a slight overhead caused by the necessity of transmitting messages containing current statuses of nodes to the coordinator and broadcasting decisions of the coordinator to all nodes in the network.

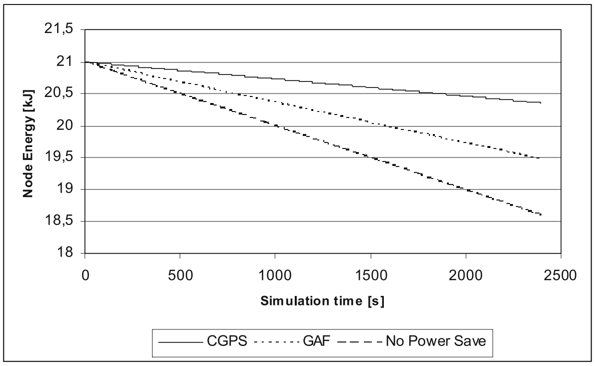


Fig. 4. Average energy consumption for transmission; network size – 60 nodes

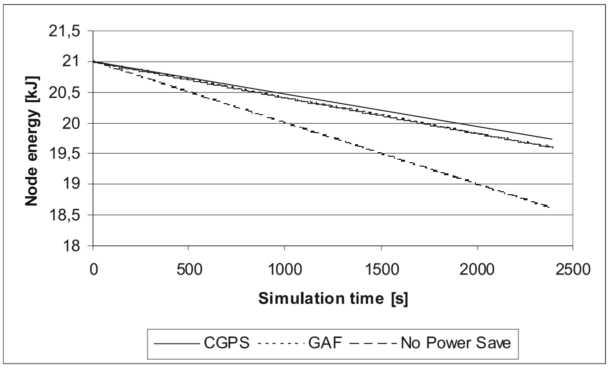


Fig. 5. Average energy consumption for transmission; network size – 90 nodes

4 Summary and Conclusions

Many challenges arise from wireless ad hoc networking and its application. We focused on one of them that is especially important in ubiquitous sensing and actuation – energy efficient communication. In our paper a brief review of some representative energy aware communication techniques was provided. We described and compared through simulation the selected energy aware protocols rely on power control and activity control. Finally, we presented and evaluated a new clustering based approach that utilizes the periodical coordination to reduce the overall energy usage by the network.

As a final observation we can say that the requirement of cooperation between power saving techniques and routing protocols is particularly useful in WSNs where nodes forward packets for each other. In practice, however it is not straightforward. Therefore, strategies for energy aware and reliable communication in WSN are discussed in the extensive literature and has became a hot debate nowadays. It is worth to note that power control and activity control are two different methods for energy efficient communication. The combination of them has not been yet well studied. The integration of these both techniques can be an interesting research topic.

Acknowledgments. The author would like to thank Izabela Windyga and Piotr Kwasniewski for assistance in preparing computational results.

References

1. Abbasi, A., Younis, M.: A survey on clustering algorithms for wireless sensor networks. Computer Communications Archive 30(14-15), 2826–2841 (2007)
2. Akyildiz, I., Vuran, M.: Wireless Sensor Networks. John Wiley & Sons, Ltd., West Sussex (2010)

3. Anastasi, G., Conti, M., Francesco, M.D., Passarella, A.: Energy conservation in wireless sensor networks: A survay. Ad Hoc Networks 7, 537–568 (2009)
4. Anderson, B., Mao, G., Fida, B.: Wireless sensor network localization techniques. Computer Networks 51(10), 2529–2553 (2007)
5. Asiam, N., Philips, W., Robertson, W., Sivakumar, S.: A multi-criterion optimization technique for energy efficient cluster formation in wireless sensor networks. Information Fusion 12(3), 202–212 (2011)
6. Bandyopadhyay, S., Coyle, E.: Minimizing communication costs in hierarchically-clustered networks of wireless sensors. Computer Networks 44(1), 1–16 (2004)
7. Blough, D.M., Leoncini, M., Resta, G., Santi, P.: The k-neighbors approach to interference bounded and symmetric topology control in ad hoc networks. IEEE Trans. on Mobile Computing 5, 1267–1282 (2006)
8. Borbash, S.A., Jennings, E.H.: Distributed topology control algorithm for multihop wireless networks. In: Proc. World Congress on Computational Intelligence (WCCI 2002), pp. 355–360 (2002)
9. Buttyan, L., Schaffer, P.: Position-based aggregator node election in wireless sensor networks. International Journal of Distributed Sensor Networks 2010, 1–15 (2010)
10. Casari, P., Marcucci, A., Nati, M., Petrioli, C., Zorzi, M.: A detailed simulation study of geographic random forwarding. In: Proc. of Military Communications Conference, MILCOM, vol. 1, pp. 59–68 (2005)
11. Chamam, A., Pierre, S.: A distributed energy-efficient clustering protocol for wireless sensor networks. Computers & Electrical Engineering 36(2), 303–312 (2010)
12. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. ACM Wireless Networks 8(5), 481–494 (2002)
13. Ghorbannia Delavar, A., Artin, J., Tajari, M.M.: PRWSN: A hybrid routing algorithm with special parameters in wireless sensor network. In: Özcan, A., Zizka, J., Nagamalai, D. (eds.) WiMo 2011 and CoNeCo 2011. CCIS, vol. 162, pp. 145–158. Springer, Heidelberg (2011)
14. Dhivya, M., Sundarambal, M.: Cuckoo search for data gathering in wireless sensor networks. International Journal of Mobile Communications 9(6), 642–656 (2011)
15. Ding, P., Holliday, J., Celik, A.: Distributed energy-efficient hierarchical clustering for wireless sensor networks. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 322–339. Springer, Heidelberg (2005)
16. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless sensor networks. In: Proc. of the 33rd Hawaii International Conference on System Sciences, pp. 1–10 (2000)
17. Ilyas, M., Mahgoub, I.: Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems. CRC Press LLC, USA (2005)
18. Karl, H., Willig, A.: Protocols and Architectures for Wireless Sensor Networks. John Wiley & Sons, Ltd., West Sussex (2005)
19. Kasch, W., Ward, J., Andrusenko, J.: Wireless network modeling and simulation tools for designers and developers. IEEE Communications Magazine 12, 120–127 (2008)
20. Kubisch, M., Karl, H., Wolisz, A., Zhong, L.C., Rabaey, J.M.: Algorithms for transmission power control in wireless sensor networks. In: IEEE WCNC (2003)
21. Levis, P.: TinyOS programming, USA (2006)
22. Li, N., Hou, J., Sha, L.: Design and analysis of an mst-based topology control algorithm. In: Proceedings of IEEE Infocom 2003, pp. 1702–1712. IEEE Computer Society, San Francisco (2003)

23. Liu, X., Shao, F., Xin, P.: Energy balance optimization for prolonging the lifetime of wireless sensor network. In: Proc. of International Conference on Multimedia Technology (ICMT), Ningbo, pp. 1–4 (2010)
24. Liu, Y., Zhang, Q., Ni, L.: Opportunity-based topology control in wireless sensor networks. IEEE Transactions on Parallel and Distributed Systems 21(3), 405–416 (2010)
25. Marks, M.: A survey of multi-objective deployment in wireless sensor networks. Journal of Telecommunications and Information Technology 3, 36–41 (2010)
26. Niewiadomska-Szynkiewicz, E., Marks, M.: Optimization schemes for wireless sensor network localization. Journal of Applied Mathematics and Computer Science 19(2), 291–302 (2009)
27. Rappapport, T.: Wireless Communications: Principles and Practic. Communications Engineering and Emerging Technologies Seriess. Prentice Hall, USA (2002)
28. Rodoplu, V., Meng, T.: Minimum energy mobile wireless networks. IEEE Journal Selected Areas on Mobile Computingg 17(8), 1333–1344 (1999)
29. Santi, P.: Topology Control in Wireless Ad Hoc and Sensor Networks. John Wiley & Sons, Ltd., West Sussex (2006)
30. Shukur, M., Chyan, L., Yap, V.: Wireless sensor networks: delay guarantee and energy efficient mac protocols. World Academy of Science, Engineering and Technology 50, 1061–1065 (2009)
31. Sikora, A., Niewiadomska-Szynkiewicz, E.: A parallel and distributed simulation of ad hoc networks. Journal of Telecommunications and Information Technology 3, 76–84 (2009)
32. Thein, M., Thein, T.: An energy efficient cluster-head selection for wireless sensor networks. In: Proc. of International Conference on Intelligent Systems, Modelling and Simulation, Liverpool, UK, pp. 287–291 (2010)
33. Verdone, R., Dardari, D., Mazzini, G., Conti, A.: Wireless Sensor Networks and Actuator Networks. Technologies, Analysis and Design. Elsevier, USA (2008)
34. Wattenhofer, R., Li, L., Bahl, P., Wang, Y.: Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In: Proc. of IEEE Infocom 2001, pp. 1388–1397 (2001)
35. Wattenhofer, R., Zollinger, A.: Xtc: A practical topology control algorithm for ad-hoc networks. In: Proc. of the 4th Inter. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks, WMAN (2003)
36. Xu, Y., Heidemann, J., Estrin, D.: Geography-informed energy conservation for ad hoc routing. In: Proceedings of the 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001), pp. 70–84. ACM, New York (2001)
37. Younis, O., Fahmy, S.: Distributed clustering in ad-hoc sensor networks: A hybrid, energy-efficient approach. In: Proc. of the IEEE INFOCOM, vol. 1, pp. 629–640 (2004)

GPU Acceleration for Hermitian Eigensystems

Michael T. Garba[1], Horacio González–Vélez[2], and Daniel L. Roach[3]

[1] IDEAS Research Institute, Robert Gordon University, Aberdeen AB25 1HG, UK
m.t.garba@rgu.ed.ac.uk
[2] Cloud Competency Centre, National College of Ireland, Dublin 1, Ireland
horacio@ncirl.ie
[3] Physics and Materials Research Centre, University of Salford, Salford M5 4WT, UK
d.roach@salford.ac.uk

Abstract. As a recurrent problem in numerical analysis and computational science, eigenvector and eigenvalue determination usually employs high-performance linear algebra libraries. This paper explores the implementation of high-performance routines for the solution of multiple large Hermitian eigenvector and eigenvalue systems on a Graphics Processing Unit (GPU). We report a performance increase of up to two orders of magnitude over the original EISPACK routines with a NVIDIA Tesla C2050 GPU, providing an effective order of magnitude increase in unit cell size or simulated resolution for Inelastic Neutron Scattering (INS) modelling from atomistic simulations.

Keywords: GPU, Eigensystems, CUDA, Parallel Computing, Computational Linear Algebra.

1 Introduction

Eigenvector and eigenvalue determination are frequently encountered problems in numerical analysis and computational science for which a number of broadly applied libraries currently exist. However, as increasingly elaborate models become of practical value to scientific and engineering applications, the demands of solving these systems typically require high performance computing with clusters or supercomputers.

As an emerging architecture for high-performance parallel computing, the Graphics Processing Unit (GPU) has the potential to enable a new generation of applications for desktop machines and small clusters. Originally intended for intensive real-time 3D graphics and gaming, GPUs have demonstrated cluster-level performance at a fraction of the cost and energy consumption of traditional CPUs for certain general purpose applications. GPU advances are expected to sustain the trend of Moore's law that conventional CPUs are straining to maintain.

However, the shift towards GPU computing is a drastic architectural change that has left a void in the space of application software and support libraries that are able to leverage the full capabilities of the platform. While the solution to computational modelling problems–which were impractical on the desktop and uneconomical on the supercomputer–may very well become the dominant GPU applications of the future, effective GPU programming remains an open problem in computational science.

With NVIDIA's CUDA, AMD's Firestream, Microsoft's DirectCompute and the vendor-neutral OpenCL platform, significant resources are being directed towards developing a supporting ecosystem for GPU computing in the form of libraries, specifications, and tools that are at various levels of maturity.

Driven by our own immediate need for high performance solvers for modelling Inelastic Neutron Scattering (INS) data for comparison and fitting with experimentally obtained data sets from central facilities such as ISIS, ILL and SNS in Europe and the rest of the world, this paper describes our application of GPUs to the solution of Hermitian eigensystems, based on the Eispack library and using NVIDIA's CUDA platform. Our work may arguably shed some light on a computational problem of even wider significance and applicability than this intended modelling application.

2 Background

Eispack, and its successor Lapack, provide extensive dense linear algebra routines applied in mathematical and scientific computing. Originally developed in the US in the seventies [16], the accuracy and numerical stability of Eispack has been established through diverse application over the past 30+ years, leading to a number of developments in the field [3].

We have developed an initial high performance parallel version of Scatter [15], a software component that introduces coherent and incoherent polycrystalline INS (poly-CINS) calculation capabilities for lattice models into the highly popular Molecular/Lattice Dynamics package, GULP, also known as the General Utility Lattice Program [6]. This application has demonstrated linear scaling up to 1024 nodes on the Huygens prototype supercomputer in the SARA facilities in the Netherlands [8]. The core functionality of the application is centred around phonon mode calculations carried out with the support of Eispack. However, not every Scatter deployment may have access to major supercomputing installations and it is clear that more affordable computational power is required on the lower end of the computing scale [1].

Conversely, GPU modules are becoming a frequent presence in high performance computing platforms and the application of Scatter to progressively more complex models on larger installations will require effective usage of these resources. As a result, an efficient GPU implementation of the most computationally intensive parts of the Scatter routine will alleviate this imperative demand. Of particular interest are solvers for the class of Hermitian eigensystems that occur in INS modelling, arising from the determination of vibratory phonon modes and their associated scattering contributions [14].

For numerically intensive tasks, GPUs have substantial computing potential [17]. However, complex control flow with conditional branching and thread divergence incur a noticeable performance penalty [13]. To achieve reasonable performance benefits, it is necessary to augment traditional development techniques with low-level knowledge of the underlying GPU architecture [10].

2.1 CUDA Platform

The Compute Unified Device Architecture (CUDA) is NVIDIA's platform for GPU computing, providing compilation tools, libraries, a runtime system and hardware specifications. CUDA allows the execution of *kernels*, written in CUDA C, on the GPU device. A kernel executes as a configurable grid of independent thread blocks that may contain up to 1024 threads in second generation CUDA devices.

A *Single Instruction Multiple Thread (SIMT)* abstraction, where threads within a block execute identical instructions and may operate on different memory locations, allows fine-grained data parallelism within thread blocks and task parallelism at kernel level where multiple blocks may execute independently [12]. Thread blocks are divided into *warps* of 32 threads in 2nd generation CUDA devices. For a given block, only one of these warps is scheduled to execute on the actual hardware at any given instant.

GPU memory is hierarchically organised and independent from host memory. Global Memory, high-latency and high-bandwidth DRAM, is the primary memory available on the device and is accessible by all executing kernels as well as for host to GPU data transfer. Limited high-speed Shared Memory, essentially a user-managed cache, exists locally on each streaming multiprocessor to allow the explicit avoidance of expensive off-chip global memory accesses. Also present are register, texture and constant memories with various performance characteristics.

The GPU architecture and best practices for achieving good performance are extensively documented in the CUDA platform and memory transfer contention and bandwidth represent the predominant bottlenecks to GPU performance. A critical performance consideration is that high-cost global memory operations can be performed simultaneously or *coalesced* for a thread warp if certain access constraints are satisfied. In practice, significant efforts are usually dedicated to optimising memory access patterns of this kind by what is frequently a hit-or-miss approach involving conflicting trade-offs to maximise the *compute to global memory access (CGMA) ratio* [10]. Different authors have suggested various approaches to optimising memory usage for scientific applications in GPU architectures, such as the use of cache analysis techniques to improve tiling algorithms [9], the deployment of low-level compiler annotations within CUDA source files to steer traversal of the memory hierarchy [19], and the automatic translation of OpenMP structures into CUDA primitives [11].

However, such techniques are both application and architecture dependent, and may rely on manual intervention for code analysis, annotation, or tool coupling. Ideally, seamless integration of the application and the architecture without the need for additional code or human intervention is highly desirable. Extensive efforts are now geared towards the use of platform-agnostic GPU frameworks which can deal with standard unified language deployments such as CUDA and OpenCL [4].

2.2 Contribution

Despite a number of emerging GPU numerical libraries, no non-proprietary library for eigensystem analysis is available to completely meet the application requirements. Therefore, a basic port of the required functional subset of EISPACK to the GPU has been undertaken. Admittedly, the more modern LAPACK–which has largely superseded EISPACK–may have formed a functionally superior basis. However, the inherent architectural complexity and reliance on an efficient BLAS implementation implies a long-term effort that the immediacy of our requirements does not allow. The MAGMA library is such an effort that is in the early stages of providing hybrid multicore-CPU/GPU implementations of LAPACK routines [18].

The challenges of achieving efficient performance on a GPU architecture may justify the extended effort of custom algorithms developed specifically for the strengths of the platform [20]. However, we maintain the original algorithms of the legacy EISPACK implementation for several reasons:

1. This work is motivated by a very practical application for which the EISPACK eigensolver has proven adequate.
2. As EISPACK has been in production use for nearly 40 years, the numerical characteristics and accuracy have been established by exhaustive application and testing.
3. The problem of creating a data-parallel GPU version is conceptually similar to that of creating a vector-processor version of the EISPACK routines. A vector implementation was created for the IBM 3090-VF [2].
4. While alternative algorithms used in LAPACK possess superior cache usage characteristics and performance in modern processor architectures, they provide this at the expense of software complexity and reliance on an efficient BLAS implementation.

The EISPACK implementation provides the `ch` driver for double-precision Hermitian matrices and its three subroutine dependencies shown in Table 1. It extends our initial work [7] by introducing a coordinated approach to the development of GPU-based eigensystem solvers.

3 Hermitian Eigensystems

For a general square matrix A, a non-zero vector $\mathbf{v}$ is an eigenvector if and only if there exists a corresponding non-zero scalar λ, or eigenvalue, which satisfies equation (1). For a *Hermitian* matrix A, a complex matrix which is equal to its conjugate transpose ($A = A^{\dagger}$), it can be demonstrated that all eigenvalues λ_i are real.

$$A\mathbf{v} = \lambda\mathbf{v} \tag{1}$$

In equation (2), T is an invertible matrix, the matrices A and B are said to be *similar* and share several important properties that include identical eigenvalues λ_i and closely related eigenvectors (equation (3)).

$$T^{-1}AT = B \tag{2}$$

$$E_A = TE_B \tag{3}$$

Similarity transformations of the form of equation (2) are the basis of several matrix algorithms. It is frequently possible to perform a reduction to a similar matrix that allows the use of more efficient or direct algorithms. The Schur decomposition of A is such a representation of the matrix as the result of a similarity transformation (equation (4)) on a strictly upper-triangular matrix S with a unitary matrix Q.

$$A = QSQ^{-1} \tag{4}$$

A transformation such as equation (4) preserves symmetry when Q is unitary i.e. $QQ^\dagger = Q^\dagger Q = I$. Thus, if A is Hermitian and S is strictly upper triangular then S must be a diagonal matrix. Since $AQ = QS$, the columns of Q must be eigenvectors of A and the diagonal entries of S are the corresponding eigenvalues. Therefore, the Hermitian eigensystem problem is equivalent to determining the Schur decomposition of the Hermitian matrix A.

To compute the Schur decomposition, the matrix is typically reduced to a similar Hessenberg matrix via a potentially infinite sequence of symmetry-preserving similarity transformations. Since an upper Hessenberg matrix has all entries below the first subdiagonal set to zero (equation (5)), another property which follows from symmetry is that the Hessenberg form of a Hermitian matrix is tridiagonal. In practice, a finite number of these elementary Householder or Givens reflections will cause off-diagonal elements to effectively reach zero as they are reduced to less than machine roundoff error.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1(n-1)} & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2(n-1)} & a_{2n} \\ 0 & a_{32} & a_{33} & \dots & a_{3(n-1)} & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{(n-1)(n-1)} & a_{(n-1)n} \\ 0 & 0 & 0 & \dots & a_{n(n-1)} & a_{nn} \end{bmatrix} \tag{5}$$

The eigenvalues and eigenvectors of the tridiagonalised matrix are evaluated via the QR algorithm. While the eigenvalues are identical, the actual eigenvectors E_A of the original matrix are retrieved by back-transforming the computed eigenvectors.

4 GPU Kernel Implementation and Optimisation

The original EISPACK library was developed as a Fortran port of a set of Algol routines developed for numerical computation in the Handbook for Automatic Computation [21]. Our intention was to gain parallel performance on the GPU architecture while preserving the original high-quality algorithmic implementation of Eispack. These Fortran routines require source-level translation into equivalent C sources for compatibility with the C/C++-based CUDA SDK and compilation tools.

The f2c tool [5] allows direct compilation of standard Fortran77 code into functionally equivalent C code with transparent handling of notable language differences such as the row-major vs. column-major array representation formats.

Table 1. Relevant Hermitian Eigensystem routines in Eispack used by the ch driver

Routine	Description
htridi	Reduction of complex Hermitian matrix to real symmetric tridiagonal matrix via unitary similarity transformations.
tql2	Eigenvalues and eigenvectors of symmetric tridiagonal matrix by QL method.
htribk	Eigenvectors of complex Hermitian matrix by back-transformation of corresponding real symmetric tridiagonal matrix.

Under the ch driver for double precision Hermitian matrices, Eispack relies on the three subroutines outlined in Table 1. These subroutines have served as the basis of three functionally equivalent GPU kernels.

The implementation is executed in a series of steps, illustrated in Figure 1, with testing and verification of sequential equivalence. For each routine, an initial *proto-kernel* is implemented without any CUDA parallel constructs to test code execution on the GPU for data transfer, kernel launch and data retrieval using a single thread. This proto-kernel is highly inefficient as it uses very little of the computational capabilities of the device.

With the execution model and platform verified, synchronisation constructs are necessary for global memory operations to avoid race conditions before the actual fine-grained work distribution can be introduced. Performance gains emerge as data-parallel operations are distributed between cooperating threads. These loops are identified from source-level line-profiling on the original CPU version of Eispack, the rationale being that CPU performance is strongly indicative of potential performance bottlenecks in the GPU kernels. This is a necessary workaround as CUDA profiling tools provide relatively basic functionality.

Multiple independent blocks provide coarse-grained block-level parallelism, allowing the GPU to solve several independent eigensystems simultaneously in a single kernel invocation. While, the CUDA platform provides the __syncthreads() primitive for thread synchronisation within a block, global kernel synchronisation across different thread blocks is unsupported.

In this implementation, a number of thread blocks independently handle the solution of multiple eigensystems in parallel. With a thread block or cooperative thread array (CTA) mapped to an input problem set, parallelism is possible at

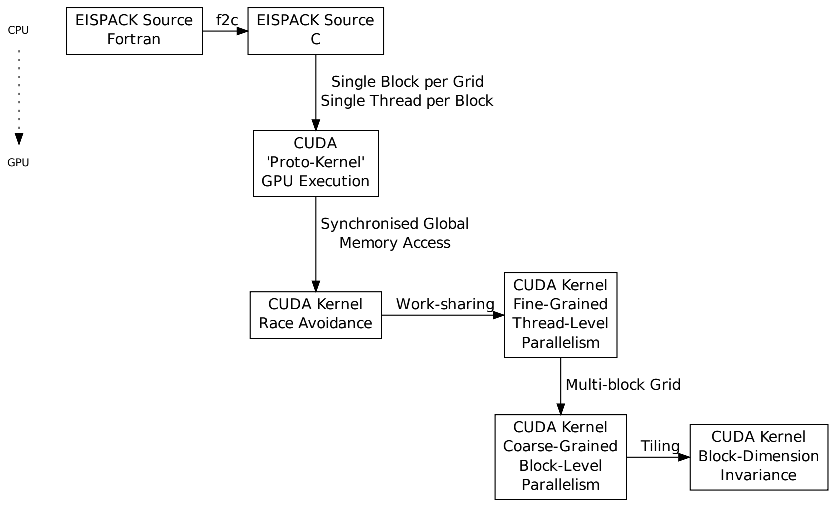

CPU
GPU
EISPACK Source Fortran
f2c
EISPACK Source C
Single Block per Grid
Single Thread per Block
CUDA 'Proto-Kernel' GPU Execution
Synchronised Global Memory Access
CUDA Kernel Race Avoidance
Work-sharing
CUDA Kernel Fine-Grained Thread-Level Parallelism
Multi-block Grid
CUDA Kernel Coarse-Grained Block-Level Parallelism
Tiling
CUDA Kernel Block-Dimension Invariance

Fig. 1. Development process with implementation steps from CPU to GPU before optimisation and performance tuning

both independent block and cooperative thread levels. Tiling allows the actual dimensions of the thread block to be completely independent of the dimensions of the matrix problem.

4.1 Optimisation

Arguably, optimisation is currently the most challenging aspect of GPU programming and, as memory transfer constitutes the predominant limit to achievable performance, the objective is usually to maximise the *Compute to Global Memory Access* (CGMA) ratio. On account of the novelty and complexity of the platform, the compilation tools do not provide the same level of optimised code generation that traditional CPUs have available. As a result, this responsibility rests with the programmer and a mental model of the hardware architecture of the GPU platform is necessary.

The CUDA platform provides extensive documentation of performance best practices [13]. However, trade-offs remain necessary between possible efficiency measures. Some performance optimisations applied include:

1. Asynchronous transfers between Host and GPU memory over multiple streams allow concurrent kernel execution and overlapped I/O.
2. Coalesced memory access by algorithm reorganisation. Transposed matrix layout in some subtasks is necessary to achieve higher memory transfer bandwidth.

3. Improved register memory usage by the elimination (or reuse when appropriate) of extraneous register variables to improve GPU occupancy and facilitate latency hiding on the streaming multiprocessors.
4. Use of explicit caching in shared memory to limit costly global memory accesses.
5. Empirical determination of launch configuration by trial and error. While, the guidelines recommend that thread blocks sizes should be multiples of a warp to allow latency hiding for multiple warps, it is necessary to determine actual optimal block sizes by testing. The different kernels performed optimally at distinct block dimensions.

5 Performance Evaluation

Performance evaluations are carried out on a 64-bit Dell Precision T7500 Server with 4 Intel Xeon 2GHz CPU cores, 4GB RAM and a NVIDIA Tesla C2050 GPU with a PCI express interface running Version 3.2 of the CUDA SDK on 64-bit Ubuntu 10.04 Linux.

The second generation NVIDIA Tesla C2050 GPU is designed specifically for scientific and numerical computing applications. 14 streaming multiprocessors (SM), each providing 32 streaming processors (SP), offer 448 parallel cores in

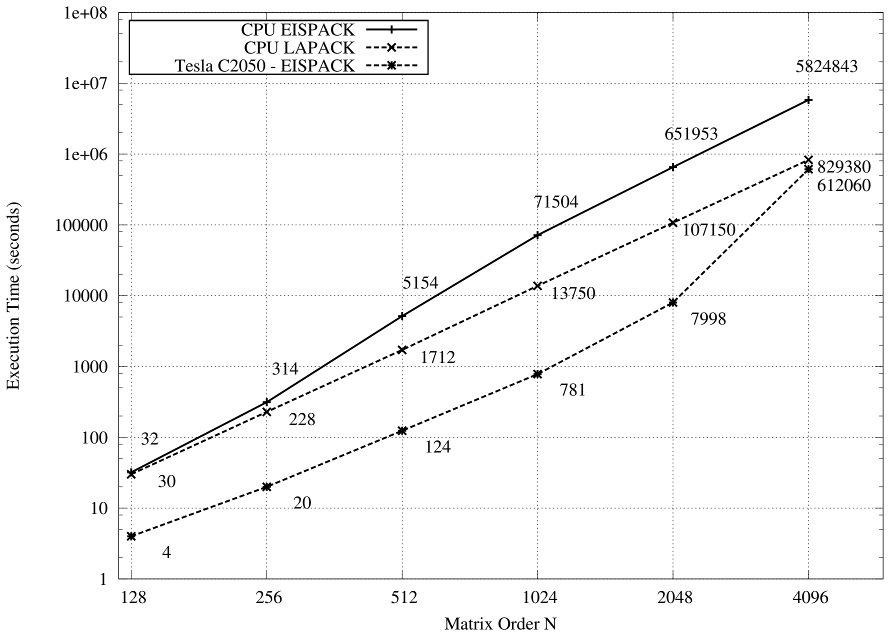


Fig. 2. Execution time for 1000 double precision Hermitian matrices of order N with (i) the current EISPACK CPU implementation, (ii) LAPACK on CPU and (iii) the test EISPACK implementation on a NVIDIA Tesla C2050 GPU.

Table 2. NVIDIA Tesla C2050 GPU Specifications

Parameter	Value
Number of CUDA Cores	448
Frequency of CUDA Cores	1.15GHz
Double Precision floating point performance (peak)	515 Gflops
Single Precision floating point performance (peak)	1.03 Tflops
Total Dedicated Memory	3GB GDDR5

total. While many earlier GPUs completely lacked double precision support, the Tesla GPU provides improved double-precision floating point performance.

The execution times for 1000 N-order input matrices with Eispack and Lapack on a single CPU core and on the GPU are shown in Figure 2. GPU times are collected via the platform timers and are inclusive of memory transfer overhead.

Within a critical window ($N = 512 - 2048$), the current GPU routine yields performance increases of between 50 to a 100 times over the previous Eispack implementation, a result of performance gains at both thread and block levels. As the matrix order increases, the GPU memory is able to accommodate fewer matrices to provide any block-level performance advantage and execution resources begin to idle. Therefore, the scalability of the approach is restricted for higher values of N by the hard limit that memory places on GPU occupancy despite the still-observable benefits of thread-level parallelism.

The superior Lapack cache behaviour delivers consistently higher performance over Eispack for larger values of N. While equivalent routines in both Lapack and Eispack are of storage order $O(n^2)$, Lapack reuses the same input matrix memory for output and is therefore more memory efficient.

6 Conclusions

The particular applicability of INS to the study of nano-materials has led to increasing popularity for structural determination in the materials science community. Libraries of mathematical routines remain the foundation of these applications and it is important to establish and maintain efficient implementations.

The intention of this work has been to create an efficient GPU port that meets the need created by Scatter and that is based on the established numerical Eispack code. We have demonstrated the substantial performance potential of the GPU in INS modelling and similar applications that rely on significant numerical computation and anticipate the emergence of standard numerical libraries for the GPU that are based on tuned algorithms oriented towards the particular strengths of the platform. A very recent release of the Magma library introduces a Hermitian eigensolver for hybrid multicore-CPU-GPU configurations that is based on an alternative divide-and-conquer algorithm. The suitability of this Lapack-based version is being evaluated. However, our initial observations indicate that Magma performance is optimised for very large values of N, outside the critical window identified previously.

While the nature of GPU computing implies that tuning and testing are usually on-going concerns, the current implementation has demonstrated a performance increase of two orders of magnitude with several INS models in Gulp, the Scatter host application [6], delivering consistent output. For the intended neutron scattering application, good performance within the critical window has been sufficient to allow an order of magnitude advance in the size, complexity or grid refinement of the INS models. Further work will investigate deployment in a multi-GPU cluster and other computationally intensive aspects of INS modelling that may benefit from GPU acceleration. This includes derivation of the dynamical matrix and nearest-neighbour search. In the long-term, it is expected that subsequent GPU models will offer improved memory characteristics, deliver higher performance and provide superior optimising compilers. Furthermore, the need for migration to Lapack as larger systems are modeled is evident.

The challenge of determining optimal parameters for launch configuration and performance tuning presents an opportunity to apply heuristic techniques. Ultimately, we seek to investigate deployment of the neutron scattering program for complex models in large dynamic GPU-accelerated heterogeneous environments and techniques for improving co-operative CPU-GPU throughput. This will combine CPU, GPU-Eispack and, prospectively, MAGMA in an adaptive framework that optimises for performance by balancing between alternative execution paths. Multiple implementations of GPU kernels, optimised for various problem scales, become a means of achieving this performance balance.

Acknowledgements. The authors would like to thank the Partnership for Advanced Computing in Europe (PRACE) for their support and grant of computing time in the SARA supercomputing facilities. The PRACE project receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528. The Authors would also like to acknowledge the support of a collaboration travel grant awarded by the STFC Collaborative Computational Project 5 (CCP5). One of the authors (Roach) would like to acknowledge the support of EPSRC (EP/G049130) in the development of the Scatter code.

The authors would like to express their gratitude to the NVIDIA Corporation for the donation through the Professor Partnership programme of the GPU Tesla equipment employed in this work. Our appreciation to Julian Gale of Curtin University (Australia) for making the Gulp source code available.

References

1. Bethel, E., van Rosendale, J., Southard, D., Gaither, K., Childs, H., Brugger, E., Ahern, S.: Visualization at Supercomputing Centers: The Tale of Little Big Iron and the Three Skinny Guys. IEEE Computer Graphics and Applications 31(1), 90–95 (2011)
2. Cline, A.K., Meyering, J.: Converting eispack to run efficiently on a vector processor. Tech. rep., Pleasant Valley Software, Austin, Texas (1991)
3. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van der Vorst, H.A.: Numerical linear algebra for high-performance computers, 2nd edn. SIAM (1998)
4. Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., Dongarra, J.: From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. Parallel Computing 38(8), 391–407 (2012)
5. Feldman, S.: A Fortran to C converter. ACM SIGPLAN Fortran Forum 9(2), 21–22 (1990)
6. Gale, J.D., Rohl, A.L.: The general utility lattice program (GULP). Molecular Simulation 29(5), 291–341 (2003)
7. Garba, M., González-Vélez, H.: Towards ad-hoc GPU acceleration of parallel eigensystem computations. In: ECMS 2011: 25th European Conference on Modelling and Simulation. ECMS, Krakow (June 2011)
8. Garba, M., González-Vélez, H., Roach, D.: Parallel computational modelling of inelastic neutron scattering in multi-node and multi-core architectures. In: 11th IEEE Int. Conf. on High Performance Computing and Communications, pp. 509–514. IEEE, Melbourne (2010)
9. Govindaraju, N.K., Larsen, S., Gray, J., Manocha, D.: A memory model for scientific algorithms on graphics processors. In: SC 2006: ACM/IEEE Conf. on Supercomputing, p. 6. IEEE, Tampa (2006)
10. Kirk, D., Wen-mei, W.: Programming massively parallel processors: A Hands-on approach. Morgan Kaufmann Publishers Inc., San Francisco (2010)
11. Lee, S., Min, S.J., Eigenmann, R.: Openmp to gpgpu: a compiler framework for automatic translation and optimization. SIGPLAN Not. 44, 101–110 (2009)
12. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue 6(2), 40–53 (2008)
13. Nvidia Corporation: NVIDIA CUDA C Programming Best Practices Guide. Manual Version 2.3, NVIDIA (2009), `http://developer.nvidia.com/` (last accessed: February 1, 2011)
14. Roach, D., Ross, K., Gale, J.D.: The application of coherent inelastic neutron scattering to the study of polycrystalline materials (2012) (in Preparation)
15. Roach, D.L., Gale, J., Ross, D.: Scatter: A New Inelastic Neutron Scattering Simulation Subroutine for GULP. Neutron News 18(3), 21–23 (2007)
16. Smith, B.T., Boyle, J.M., Dongarra, J., Garbow, B.S., Ikebe, Y., Klema, V.C., Moler, C.B.: Matrix Eigensystem Routines - EISPACK Guide, 2nd edn. LNCS, vol. 6. Springer, Heidelberg (1976)
17. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Computing 36(5-6), 232–240 (2010)

18. Tomov, S., Nath, R., Ltaief, H., Dongarra, J.: Dense linear algebra solvers for multicore with GPU accelerators. In: IPDPS 2010 Workshops, pp. 1–8. IEEE, Atlanta (2010)
19. Ueng, S.-Z., Lathara, M., Baghsorkhi, S.S., Hwu, W.-m.W.: CUDA-lite: Reducing GPU programming complexity. In: Amaral, J.N. (ed.) LCPC 2008. LNCS, vol. 5335, pp. 1–15. Springer, Heidelberg (2008)
20. Vázquez, F., Fernández, J.J., Garzón, E.M.: A new approach for sparse matrix vector product on NVIDIA GPUs. Concurrency and Computation: Practice and Experience 23(8), 815–826 (2011)
21. Wilkinson, J., Reinsch, C.: Linear Algebra. Handbook for Automatic Computation, vol. 2. Springer (1971)

Scalable and High Performing Learning and Mining in Large-Scale Networked Environments: A State-of-the-art Survey

Evis Trandafili[1] and Marenglen Biba[2]

[1] Department of Computer Science,
Polytechnic University of Tirana, Albania
etrandafili@fti.edu.al
[2] Department of Computer Science,
University of New York in Tirana, Albania
marenglenbiba@unyt.edu.al

Abstract. Scalability is a major issue in the application of machine learning and data mining to large-scale networked environments. While there has been important progress in the learnability of models for medium-sized datasets, there is still much challenge in facing large-scale systems. In particular, with the evolution of distributed and networked environments, the complexity of the learning and mining process has now grown due to the possibility to integrating more data in the learning process. This paper provides a survey on the state-of-the-art on the methods and algorithms to enhance scalability of machine learning and data mining for large-scale networked systems.

1 Introduction

Machine Learning and Data Mining have long dealt with the problem of inferring models for classification in many application domains. With the fast growing amount of available data, however, the capability of traditional approaches to learn useful models has reached the limit. Large networked environments are continuously posing new challenges to learning algorithms which have now to take into consideration the presence of many entities distributed in networked systems. The possibility to involve in the learning process huge collections of documents and large databases, has led to new opportunities for discovering important relationships among apparently distant entities, but at the same time, has raised performance issues that the currect machine learning methods have to deal with.

In this paper we present a state-of-the-art survey of the most important works regarding scalability and performance of machine learning and data mining approaches for large-scale networked data. The paper is organized as follows: in Section 2 we present research regarding mining of large social networks and how data mining approaches have tackled the problem of scalability in these networks. Section 3 presents research in sensor networks outlining the major contributions in scaling machine learning in this kind of environments. In Section 4 we present research on high performing algorithms in P2P systems. Section 5 presents research activity and results for distributed control systems and stream mining and Section 6 focuses on learning and mining in large scale multimedia retrieval systems. Finally, we conclude in Section 7.

2 Scalability and Learning in Large-Scale Social Networks

With the wide use of the Internet, now complex interactions easily take place among different entities, leading to huge information networks. When these entities consist of people that interact in large online social websites, the amount of daily interactions grows rapidly leading to large-scale social networks that need to be analyzed. Social networks are increasingly inspiring research in machine learning and data mining. This is due to the growing amount of available data that have to be analyzed. Moreover, most social networks have an outstanding marketing value and developing methods for viral marketing is a hot topic in the research community [14].

However, most social networks remain impossible to be fully analyzed and understood due to prohibiting sizes and the incapability of traditional machine learning and data mining approaches to deal with this new dimension in the learning process which is the large-scale enviroment where the data are produced.

Mobile devices and wireless technologies have led to mobile social network systems which are becoming increasingly popular. In this kind of social network, the information and influence are spread in the form of word-of-mouth. For this reason it is essential to search for a subset of influential individuals in a mobile social network such that targeting them initially (e.g. for marketing campaigns) will maximize the spread of the influence. However, unfortunately, it has been shown that the problem of finding the most influential nodes is NP-hard. It has also been shown that a Greedy algorithm with provable approximation guarantees can provide good performance, but it is computationally expensive, if not prohibitive, to run the greedy algorithm on a large mobile network. Therefore the scalability problem is present and needs to be handled appropriately.

In [64] the authors propose a new algorithm called Community-based Greedy algorithm for mining top-K influential nodes. The proposed algorithm encompasses two components: 1) an algorithm for detecting communities in a social network; and 2) a dynamic programming algorithm for selecting communities to find influential nodes. Empirical experiments on a large real-world mobile social network show that their algorithm is more than an order of magnitudes faster than the state-of-the-art Greedy algorithm for finding top-K influential nodes and the error of their approximate algorithm is small.

Social interactions that occur regularly, will typically correspond to significant yet often infrequent and hard to detect interaction patterns. To identify such regular behavior, the authors in [27] propose a new mining problem of finding periodic or near periodic subgraphs in dynamic social networks where scalability is also a major issue. They propose a practical, efficient and scalable algorithm to find such subgraphs that takes imperfect periodicity into account and demonstrate the applicability of their approach on several real-world networks and extract meaningful and interesting periodic interaction patterns.

Social networks often involve multiple relations simultaneously. People usually construct an explicit social network by adding each other as friends, but they can also build implicit social networks through daily actions like commenting on posts, or tagging photos. The authors in [16] address this problem: given a real social networking system which changes over time, do daily interactions follow any pattern? They model the formation and co-evolution of multi-modal networks proposing an approach that discovers temporal patterns in peoples social interactions. They show the effectiveness of the approach on two real datasets (Nokia FriendView and Flickr) with 100,000 and 50,000,000 records respectively, each of which corresponds to a different social service, and spans up to two years of activity.

One solution to dealing with continuously growing social networks is compressing them so that we can substantially facilitate mining and the advanced analysis of large social networks. The optimal solution would be to compress social networks in a way that they still can be queried efficiently without decompression. For example, we should still be able to perform neighbor queries efficiently (which search for all neighbors of a query vertex), as these are the most essential operations on social networks. The problem has been addressed in [40] where the authors propose a social network compression approach based on a novel Eulerian data structure using multi-position linearizations of directed graphs. Their approach seems to be the first that can answer both out-neighbor and in-neighbor queries in sublinear time and they verfiy their design with an extensive empirical study on more than a dozen benchmark real data sets.

An important task often essential in some social networks is the discovery of communities. Usually, the given scenario is the one where communities need to be discovered with only reference to the input graph. However, for many interesting applications one is interested in finding the community formed by a given set of nodes. In [50] the authors study a query-dependent variant of the community-detection problem, which they call the community-search problem: given a graph G, and a set of query nodes in the graph, the goal is to find a subgraph of G that contains the query nodes and is densely connected. A measure of density is proposed based on minimum degree and distance constraints, and an optimum greedy algorithm is developed for this measure. The authors characterize a class of monotone constraints and they generalize the algorithm to compute optimum solutions satisfying any set of monotone constraints. Finally they modify the greedy algorithm and present two heuristic algorithms that find communities of size no greater than a specified upper bound. The experimental evaluation on real datasets demonstrates the efficiency of the proposed algorithms and the quality of the solutions they obtain.

In [6], the authors propose a scalable framework for modeling competitive diffusion in social networks. In social networks, multiple phenomena often diffuse in competition with one another. Applications of this kind include, for instance, eventual results from multiple competing diffusion models (e.g. what is the likely number of sales of a given product). The authors in [6] define the most probable interpretation (MPI) problem which technically formalizes this need. They develop algorithms to efficiently solve MPI and show experimentally that their algorithms work on graphs with millions of vertices.

An interesting work is presented in [28] where the authors instead of defining a procedure to extract sets of nodes from a graph and then attempt to interpret these sets as real communities, employ approximation algorithms for the graph partitioning problem to characterize as a function of size the statistical and structural properties of partitions of graphs that could plausibly be interpreted as communities. In addition, they define the network community profile plot, which characterizes the best possible community – according to the conductance measure – over a wide range of size scales. The authors perform a study over 100 large real–world networks, ranging from traditional and on–line social networks, to technological and information networks and web graphs, and ranging in size from thousands up to tens of millions of nodes.

On large-scale networks, there is a need to perform aggregation operations. Unfortunately the existing implementation of aggregation operations on relational databases does not guarantee superior performance in network space, especially when it involves edge traversals and joins of gigantic tables. In [57], the authors investigate the neighborhood aggregation queries: Find nodes that have top-k highest aggregate values over their h-hop neighbors. While these basic queries are common in a wide range of search and recommendation tasks, surprisingly they have not been dedicated much attention. The work in [57] proposes a Local Neighborhood Aggregation framework, to answer these queries efficiently. The approach exploits two properties unique in network space: First, the aggregate value for the neighboring nodes should be similar in most cases; Second, given the distribution of attribute values, it is possible to estimate the upper-bound value of aggregates. These two properties inspire the development of novel pruning techniques, forward pruning using differential index and backward pruning using partial distribution. Empirical results show that the proposed approach could outperform the baseline algorithm up to 10 times in real-life large networks.

3 Scaling Machine Learning in Sensor Networks

Sensor networks often present large amounts of data spread over many physically distributed nodes. Machine learning and data mining techniques have the potential to deal with these kind of data. Due to the complexity of heterogeneous networked data, important challenges have arisen such as the need for run-time data aggregation, parallel computing, and distributed hypothesis formation [8].

One of the existing approaches in sensor networks is presented in [61] where the authors present an algorithm for finding distributed icebergs-elements that may have low frequency at individual nodes but high aggregate frequency (this is a problem that arises commonly in practice). The work in [7] addresses a major challenge in data mining applications where the full information about the underlying processes, such as sensor networks or large online databases, cannot be practically obtained due to physical limitations such as low bandwidth or memory, storage, or computing power. They propose a framework for detecting anomalies from these large-scale data mining applications where the full information cannot be practically obtained.

In [49] it is presented another approach for network management in large-scale randomly-deployed sensor networks, called Energy Map, which explores the inherent relationships between the energy consumption and the sensor operation. Through nonlinear manifold learning algorithms the approach visualizes the residual energy level of each sensor in a large scale network, infers the sensor locations and the current network topology through mining the collected residual energy data in a randomly-deployed sensor network, and explores the inherent relation between sensor operation and energy consumption to find the dynamic patterns from large volumes of sensor network data for network design.

In [44] and [45] the author proposes a declarative query language and data mining techniques to discover frequent event patterns and their spatial and temporal properties. In these works, raw streams of sensor readings are collected for later offline processing and analysis and in-network data mining techniques are explored to discover frequent event patterns and their spatial and temporal properties.

The authors of [35] propose and evaluate distributed algorithms for data clustering in self-organizing ad-hoc sensor networks with computational, connectivity, and power constraints. One of the benefits of in-network data clustering algorithms is the capability of the network to transmit only relevant, high level information, namely models, instead of large amounts of raw data, also reducing drastically energy consumption. Finally, the work in [43] presents an exploration of different characteristics of sensor networks which define new requirements for knowledge discovery, with the common goal of extracting some kind of comprehension about sensor data and sensor networks, focusing on clustering techniques which provide useful information about sensor networks as it represents the interactions between sensors.

In [32], the authors propose a distributed traffic stream mining system. The central server performs various data mining tasks only in the training and updating stage and sends the interesting patterns to the sensors. The sensors monitor and predict the coming traffic or raise alarms independently by comparing the data with the patterns observed in the historical streams. The sensors provide real-time response with less wireless communication and small resource requirement, and the computation burden on the central server is reduced. The authors evaluate the system on the real highway traffic streams in the GCM Transportation Corridor in Chicagoland.

4 Scalability in Peer-to-Peer Networks

The research in machine learning and data mining on analyzing data in Peer-to-Peer (P2P) networks is attracting a lot of attention of researchers. We will briefly describe here some recent developments in this exciting area.

An important work is presented in [4] where it is proposed a local distributed algorithm for expectation maximization in large peer-to-peer environments. The proposed algorithm can be used for a variety of well-known data mining tasks in a distributed environment such as clustering, anomaly detection, or target tracking. This technology is crucial for many emerging P2P applications for bioinformatics, astronomy, social networking, sensor networks and web mining. The distributed algorithm was shown provably-correct i.e. it converges to the same result compared to a similar centralized algorithm and can automatically adapt to changes to the data and the network. The authors showed also that the communication overhead of the algorithm is very low due to its local nature. This monitoring algorithm is then used as a feedback loop to sample data from the network and rebuild the model when it is outdated. Thorough experimental results were presented by the authors to verify the theoretical claims.

Tagging information is often an important feature to exploit in analyzing text documents. In many application areas involving classification of text documents, web users participate in the tagging process and the collaborative tagging results in the formation of large scale P2P systems which can function, scale and self-organize in the presence of highly transient population of nodes and do not need a central server for co-ordination. In [17] it is presented a P2P classifier learning system for extracting patterns from text data where the end users can participate both in the task of labeling the data and building a distributed classifier on it. The approach is based on a novel distributed linear programming based classification algorithm which is asynchronous in nature. The authors provide extensive empirical results on text data obtained from the online repository of NSF Abstracts Data.

An important challenge in data mining over P2P networks is the right data representation. In [67] the authors describe an approach to collaborative feature extraction, selection and aggregation in distributed, loosely coupled domains. The authors focus on scenarios in which a large number of loosely coupled nodes apply data mining to different, usually very small and overlapping, subsets of the entire data space. The goal is to learn a set of local concepts and not to find a global concept. The paper proposes two models for collaborative feature extraction, selection and aggregation for supervised data mining. One is based on a centralized P2P architecture, and the other on a fully distributed P2P architecture. The comparison of both models is performed on a real word data set.

In [24] the authors propose a local distributed algorithm for multivariate regression in large P2P environments. The algorithm was designed for distributed inferencing, data compression, data modeling and classification tasks in emerging P2P applications for bioinformatics, astronomy, social networking, sensor networks and web mining. The proposed approach proceeds in two steps. First, it offers an efficient local distributed algorithm that monitors the quality of the current regression model. If the model is outdated, it uses this algorithm as a feedback mechanism for rebuilding the model. Experimental results support the theoretical claims.

In [25], the authors propose a scalable, local privacy-preserving algorithm for distributed peer-to-peer data aggregation useful for many advanced data mining/ analysis tasks such as average/sum computation, decision tree induction, feature selection, and more. The proposed approach works in an asynchronous manner through local interactions and it was shown to be highly scalable. It particularly deals with the distributed computation of the sum of a set of numbers stored at different peers in a P2P network in the context of a P2P web mining application.

One particular type of systems are those based on the publish/subscribe model that have been adopted by many services to deliver data between distributed users based on application-specific semantics. In such systems, the semantic expressiveness of content matching and the scalability of the matching mechanism, are often found to be in conflict due to the complexity associated with content matching. In [59], the authors present a novel content-based publish/subscribe architecture based on P2P matching trees. The system achieves scalability by partitioning the responsibility of event matching to self-organized peers while allowing customizable matching functionalities. Experimental results using a variety of real world datasets demonstrate the scalability and flexibility of the system.

5 Scaling in Distributed Control Systems and Stream Mining

Machine learning and data mining provide excellent methods and techniques for dealing with automation and control in a distributed setting. Here we explore some approaches that have proven successful in important areas such as transportation, fleets and automation control.

In [23] it is presented a distributed vehicle performance data mining system designed for commercial fleets. The MineFleet system analyzes high throughput data streams onboard the vehicle, generates the analytics, sends them to the remote server over the wide-area wireless networks and offers them to the fleet managers using stand-alone and web-based user-interface. MineFleet is probably one of the first commercially successful distributed data stream mining systems. Another approach was proposed in [33] called mobility-based clustering that deals with practical research on hot spots in smart city taking into consideration unique features, such as highly mobile environments, supremely limited size of sample objects, and the non-uniform, biased samples. The authors report performance of mobility-based clustering based on real traffic situations.

The authors in [62] deal with the critical problem in a crisis situation of how to efficiently discover, collect, organize, search and disseminate real-time disaster information. The proposed system exploits the latest advances in data mining technologies to analyze the integrated input data from different sources. Another interesting approach is that in [18] where a massive quantity of complex, dynamic, and distributed location traces is handled and mined to provide effective mobile sequential recommendation.

In another recent work, a novel approach was presented based on the theory of multiple kernel learning to detect potential safety anomalies in very large data bases of discrete and continuous data from world-wide operations of commercial fleets [12]. Their results show that the proposed algorithm uncovers operationally significant events in high dimensional data streams in the aviation industry which are not detectable using state of the art methods. Another interesting approach is that of [26] where it is presented a system based on Ubiquitous Data Mining (UDM) concepts. It merges and analyses different types of information from crash data and physiological sensors to diagnose driving risks in real time.

An important feature of networked data is their uncertainty since sensors are typically expected to have considerable noise in their readings because of inaccuracies in data retrieval, transmission, and power failures. In [2] the authors propose a method for clustering uncertain data streams.

In [5] the authors propose a distributed algorithm for monitoring the principal components (PCs) for next generation of astronomy petascale data pipelines such as the Large Synoptic Survey Telescopes. This telescope takes repeat images of the night sky every 20 seconds, thereby generating 30 terabytes of calibrated imagery every night that will need to be co-analyzed with other astronomical data stored at different locations around the world. Event detection, classification and isolation in such data sets may provide useful insights to unique astronomical phenomenon. Performing data mining for such high-throughput distributed data streams is a challenging problem. In [5] it is proposed a highly scalable and distributed asynchronous algorithm for monitoring the principal components (PC) of such dynamic data streams. The authors discuss a prototype webbased system PADMINI (Peer to Peer Astronomy Data Mining) which implements this algorithm for use by the astronomers. The paper demonstrates the algorithm on a large set of distributed astronomical data to accomplish well-known astronomy tasks such as measuring variations in the fundamental plane of galaxy parameters. The proposed algorithm is provably correct (i.e. converges to the correct PCs without centralizing any data) and can seamlessly handle changes to the data or the network. Real experiments performed on Sloan Digital Sky Survey catalogue data show the effectiveness of the algorithm.

6 Scaling in Multimedia Retrieval Systems

Video and image retrieval has become an important research trend in the machine learning and data mining community. Although significant achievements have been made in this area, there are still important open challenges.

An important recent work is that in [10] where the authors propose a randomized data mining method that finds clusters of spatially overlapping images. The essential part of the proposed method relies on the min-Hash algorithm for fast detection of pairs of images with spatial overlap, the so-called cluster seeds. The seeds are then used as visual queries to obtain clusters which are formed as transitive closures of sets of partially overlapping images that include the seed. The authors show that the probability of finding a seed for an image cluster rapidly increases with the size of the cluster. The properties and performance of the algorithm are demonstrated on datasets with $10^4, 10^5$, and $5 * 10^6$ images.

The speed of the method depends on the size of the database and on the number of clusters. The first stage of seed generation is close to linear for databases sizes up to approximately 2^{34} or around 10^{10} images. On a single 2.4GHz PC, the clustering process took only 24 minutes for a standard database of more than hundred thousand images, i.e. only 0.014 seconds per image.

An important problem in multimedia content analysis and retrieval is multi-concept learning. It connects two key components in the multimedia semantic ecosystem: multimedia lexicon and semantic concept detection. In [52], the authors attempt to answer two questions related to multi-concept learning: does a large-scale lexicon help concept detection? How many concepts are enough? They perform a study on a large scale lexicon that shows that more concepts indeed help improve detection performance. The experiments show that the gain is statistically significant with more than 40 concepts and saturates at over 200. The authors also compare a few different modeling choices for multi-concept detection: generative models such as Naive Bayes perform robustly across lexicon choices and sizes, discriminative models such as logistic regression and SVM perform comparably on specially selected concept sets, yet tend to overfit on large lexicons.

Supervised ranking methods have been successfully applied in various information retrieval tasks. Among the existing methodologies, the Ranking Support Vector Machines (Rank SVMs) are well investigated. However, one major fact limiting their applications is that Ranking SVMs need optimize a margin-based objective function over all possible document pairs within all queries on the training set. In consequence, Ranking SVMs need select a large number of support vectors among a huge number of support vector candidates. In [36], the authors introduce a new model of Ranking SVMs and develop an efficient approximation algorithm, which decreases the training time and generates much fewer support vectors. Empirical studies on synthetic data and content-based image/video retrieval data show that the proposed method is comparable to Ranking SVMs in accuracy, but uses much fewer ranking support vectors and significantly less training time.

With the exponential growth of the Web, an important aspect has become near-duplicate video retrieval. Though various approaches have been proposed to address the problem, they mainly focus on the retrieval accuracy and remain infeasible to query on Web scale video database in real time. The work in [48] proposes a novel method to address the efficiency and scalability issues for near-duplicate Web video retrieval. The authors introduce a compact spatiotemporal feature to represent videos and construct an efficient data structure to index the feature to achieve real-time retrieving performance. This novel feature leverages relative gray-level intensity distribution within a frame and temporal structure of videos along frame sequence. To demonstrate the effectiveness and efficiency of the proposed method, the authors evaluate its performance on an open Web video data set containing about 10K videos and compare it with four existing methods in terms of precision and time complexity. They also test the method on a data set containing about 50K videos and 11M key-frames. It takes on average 17ms to execute a query against the whole 50K Web video data set.

A related approach is proposed in [30] where the authors propose a fast video copy– location method that can find the location of a given query in a large video repository in real time, regardless of the query length. The method proposed by the authors consists of two major contributions. First, it includes a probabilistic model for video copy location to formulate the task as a likelihood maximization problem. Second, it includes a simplified approach to reduce the maximization problem into a set of 0-1 value problems based on the indexing structure.

Another important issue of large-scale multimedia retrieval is how to develop an effective framework for ranking the search results. This problem is very challenging for content-based video retrieval due to some issues such as short text queries, insufficient sample learning, fusion of multimodal contents, and large-scale learning with huge media data. An interesting approach for this problem is proposed in [22], where the authors propose a novel multimodal and multilevel ranking framework to attack the challenging ranking problem of content-based video retrieval. The authors represent the video retrieval task by graphs and suggest a graph based semi-supervised ranking scheme, which can learn with small samples effectively and integrate multimodal resources for ranking smoothly. To make the semi-supervised ranking solution practical for large-scale retrieval tasks, they propose a multilevel ranking framework that unifies several different ranking approaches in a cascade fashion. The experiments have been performed for automatic search tasks on the benchmark testbed of TRECVID2005. The promising empirical results show that the ranking solutions are effective and very competitive with the state-of-the-art solutions in the TRECVID evaluations.

In [37] the authors propose an efficient method for face retrieval in large video datasets. In order to make the face retrieval robust, the faces of the same person appearing in individual shots are grouped into a single face track by using a reliable tracking method. The retrieval is done by computing the similarity between face tracks in the databases and the input face track. For each face track, the authors select one representative face and the similarity between two face tracks is the similarity between their two representative faces. The representative face is the mean face of a subset selected from the original face track. In this way, the authors can achieve high accuracy in retrieval while maintaining low computational cost. The experiments have been performed by extracting approximately 20 million faces from 370 hours of TRECVID video, a scale never addressed by previous attempts. The results evaluated on a subset consisting of 457,320 manually annotated faces show that the proposed method is effective and scalable.

An interesting related work is presented in [65], where the authors explore computer vision applications of the MapReduce framework that are relevant to the data mining community. The paper provides an overview of MapReduce and common design patterns are provided for those with limited MapReduce background. In addition to discussing both the high level theory and the low level implementation for several computer vision algorithms such as classifier training, sliding windows, clustering, bag-of-features, background subtraction, and image registration, the paper provides interesting experimental results for the k-means clustering and single Gaussian background subtraction algorithms are performed on a 410 node Hadoop cluster.

7 Conclusion

This paper presents a state-of-the-art in machine learning and data mining for large-scale networked environments. Recent work in large-scale mining in social networks, sensor networks, P2P systems, distributed control systems, stream databases and multimedia retrieval systems has been reviewed outlining the challenges for research in these areas. While the current state-of-the-art survey shows that relevant research has been done in dealing with scalability, there is still much room for further research in order to deal with real-world large-scale networked environments.

References

1. Agarwal, N., Liu, H., Subramanya, S., Salerno, J.J., Yu, P.S.: Connecting Sparsely Distributed Similar Bloggers. In: Proc. of Ninth IEEE International Conference on Data Mining, pp. 11–20 (2009)
2. Aggarwal, C., Yu, P.: A framework for clustering uncertain data streams. In: Proc. of 24th International Conference on Data Engineering, Cancún, México (2008)
3. Ang, H.H., Gopalkrishnan, V., Ng, W.K., Hoi, S.: On classifying drifting concepts in P2P networks. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part I. LNCS, vol. 6321, pp. 24–39. Springer, Heidelberg (2010)
4. Bhaduri, K., Srivastava, A.N.: A Local Scalable Distributed Expectation Maximization Algorithm for Large Peer-to-Peer Networks. In: Proc. of Ninth IEEE International Conference on Data Mining, pp. 31–40 (2009)
5. Bhaduri, K., Das, K., Giannella, C., Mahule, T., Kargupta, H.: Scalable, asynchronous, distributed eigen monitoring of astronomy data streams. Statistical Analysis and Data Mining 4(3), 336–352 (2011)
6. Broecheler, M., Shakarian, P., Subrahmanian, V.S.: A Scalable Framework for Modeling Competitive Diffusion in Social Networks. In: Proceedings of the 2010 IEEE Second International Conference on Social Computing, SocialCom / IEEE International Conference on Privacy, Security, Risk and Trust, PASSAT 2010, pp. 295–302 (2010)
7. Budhaditya, S., Pham, D., Lazarescu, M., Venkatesh, S.: Effective Anomaly Detection in Sensor Networks Data Streams. In: Proc. of Ninth IEEE International Conference on Data Mining, pp. 722–727 (2009)
8. Cantoni, V., Lombardi, L., Lombardi, P.: Challenges for Data Mining in Distributed Sensor Networks. In: Proc. of 18th International Conference on Pattern Recognition (ICPR 2006), vol. 1, pp. 1000–1007 (2006)
9. Chen, T., Zhong, S.: Privacy-preserving backpropagation neural network learning. IEEE Transactions on Neural Networks 20(10), 1554–1564 (2009)

10. Chum, O., Matas, J.: Large-Scale Discovery of Spatially Related Images. IEEE Trans. Pattern Anal. Mach. Intell. 32(2), 371–377 (2010)
11. Das, S., Egecioglu, O., Abbadi, A.E.: Anonymizing weighted social network graphs. In: Proc. of IEEE 26th International Conference on Data Engineering (ICDE), pp. 904–907 (2010)
12. Das, S., Matthews, B.L., Srivastava, A.N., Oza, N.C.: Multiple kernel learning for heterogeneous anomaly detection: algorithm and aviation safety case study. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)
13. Doganay, M.C., Pedersen, T.B., Saygin, Y., Savas, E., Levi, A.: Distributed privacy preserving k-means clustering with additive secret sharing. In: Proc. of the 2008 International Workshop on Privacy and Anonymity in Information Society, Nantes, France, March 29-29 (2008)
14. Domingos, P.: Mining Social Networks for Viral Marketing. IEEE Intelligent Systems 20(1), 80–82 (2005)
15. Domingos, P.: Structured Machine Learning: Ten Problems for the Next Ten Years. Machine Learning 73, 3–23 (2008)
16. Du, N., Wang, H., Faloutsos, C.: Analysis of large multi-modal social networks: Patterns and a generator. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part I. LNCS, vol. 6321, pp. 393–408. Springer, Heidelberg (2010)
17. Dutta, H., Zhu, X., Mahule, T., Kargupta, H., Borne, K., Lauth, C., Holz, F., Heyer, G.: TagLearner: A P2P Classifier Learning System from Collaboratively Tagged Text Documents. In: Proc. of Ninth IEEE International Conference on Data Mining Workshops, pp. 495–500 (2009)
18. Ge, Y., Xiong, H., Tuzhilin, A., Xiao, K., Gruteser, M., Pazzani, M.: An energy-efficient mobile recommender system. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA (2010)
19. Getoor, L., Taskar, B.: Introduction to statistical relational learning. MIT Press (2007)
20. He, J., Dai, X., Zhao, P.X.: Mixture Model Adaptive Neural Network for Mining Gene Functional Patterns From Heterogenous Knowledge Domains. International Journal of Information Technology and Intelligent Computing (2007)
21. He, D., Parker, D.S.: Topic dynamics: an alternative model of bursts in streams of topics. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)
22. Hoi, S., Lyu, M.: A Multimodal and Multilevel Ranking Scheme for Large-Scale Video Retrieval. IEEE Transactions on Multimedia 10(4), 607–619 (2008)
23. Kargupta, H., Sarkar, K., Gilligan, M.: MineFleet®: an overview of a widely adopted distributed vehicle performance data mining system. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA (2010)
24. Bhaduri, K., Kargupta, H.: An efficient local Algorithm for Distributed Multivariate Regression in Peer-to-Peer Networks. In: Peer-to-Peer Computing, pp. 212–221 (2009)
25. Das, K., Bhaduri, K., Kargupta, H.: Multi-objective Optimization Based Privacy Preserving Distributed Data Mining in Peer-to-peer Networks. Peer-to-Peer Networking and Applications 4(2), 192–209 (2011)

26. Krishnaswamy, S., Loke, S.W., Rakotonirainy, A., Horovitz, O., Gaber, M.M.: Towards Situation-awareness and Ubiquitous Data Mining for Road Safety: Rationale and Architecture for a Compelling Application. In: Proc. of Conference on Intelligent Vehicles and Road Infrastructure (IVRI 2005), University of Melbourne, February 16-17 (2005)
27. Lahiri, M., Berger-Wolf, T.Y.: Mining Periodic Behavior in Dynamic Social Networks. In: Proc. of the 8th IEEE International Conference on Data Mining (ICDM 2008), Pisa, Italy, December 15-19. IEEE Computer Society (2008)
28. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1), 29–123 (2009)
29. Lin, C.X., Zhao, B., Mei, Q., Han, J.: PET: a statistical model for popular events tracking in social communities. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)
30. Liu, B., Li, Z., Yang, L., Wang, M., Tian, X.: Real-Time Video Copy-Location Detection in Large-Scale Repositories. IEEE MultiMedia 18(3), 22–31 (2011)
31. Liu, K., Kargupta, H., Ryan, J.: Random Projection-Based Multiplicative Data Perturbation for Privacy Preserving Distributed Data Mining. IEEE Transactions on Knowledge and Data Engineering 18(1), 92–106 (2006)
32. Liu, Y., Choudhary, A.K., Zhou, J., Khokhar, A.: A Scalable Distributed Stream Mining System for Highway Traffic Data. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 309–321. Springer, Heidelberg (2006)
33. Liu, S., Liu, Y., Ni, L.M., Fan, J., Li, M.: Towards mobility-based clustering. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA (2010)
34. Liu, K., Terzi, E.: A Framework for Computing the Privacy Scores of Users in Online Social Networks, pp.288-297. In: Proc. of the Ninth IEEE International Conference on Data Mining, pp. 932–937 (2009)
35. Lodi, S., Monti, G., Moro, G., Sartori, C.: Peer-to-Peer Data Clustering in Self-Organizing Sensor Networks. In: Intelligent Techniques for Warehousing and Mining Sensor Network Data, pp. 179–212. IGI Global (2010)
36. Luo, D., Huang, H.: Ball Ranking Machines for Content-Based Multimedia Retrieval. In: Walsh, T. (ed.) IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, pp. 1390–1395. IJCAI/AAAI (2011)
37. Nguyen, T.N., Ngo, T.D., Le Sh, D.: Satoh, B. H. Le, D. A. Duong. An efficient method for face retrieval from large video datasets. In: Li, S., Gao, X., Sebe, N. (eds.) Proceedings of the 9th ACM International Conference on Image and Video Retrieval, CIVR 2010, pp. 382–389. ACM (2010)
38. Magkos, E., Maragoudakis, M., Chrissikopoulos, V., Gritzalis, S.: Accurate and large-scale privacy-preserving data mining using the election paradigm. Data and Knowledge Engineering 68(11), 1224–1236 (2009)
39. Marinai, S., Fujisawa, H. (eds.): Machine Learning in Document Analysis and Recognition. SCI, vol. 90. Springer, Heidelberg (2008)
40. Maserrat, H., Pei, J.: Neighbor query friendly compression of social networks. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)

41. Morchen, F., Dejori, M., Fradkin, D., Etienne, J., Wachmann, B., Bundschus, M.: Anticipating annotations and emerging trends in biomedical literature. In: Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27. ACM (2008)
42. Qiu, J., Lin, Z., Tang, C., Qiao, S.: Discovering Organizational Structure in Dynamic Social Network. In: Proc. of the Ninth IEEE International Conference on Data Mining, pp. 932–937 (2009)
43. Rodrigues, P.P., Gama, J., Lopes, L.: Knowledge Discovery for Sensor Network Comprehension. In: Intelligent Techniques for Warehousing and Mining Sensor Network Data, pp. 179–212. IGI Global (2010)
44. Römer, K.: Discovery of frequent distributed event patterns in sensor networks. In: Verdone, R. (ed.) EWSN 2008. LNCS, vol. 4913, pp. 106–124. Springer, Heidelberg (2008)
45. Romer, K.: Distributed Mining of Spatio-Temporal Event Patterns in Sensor Networks. In: EAWMS / DCOSS 2006, pp. 103–116, San Francisco, USA (June 2006)
46. Roth, M., Ben-David, A., Deutscher, D., Flysher, G., Horn, I., Leichtberg, A., Leiser, N., Matias, Y., Merom, R.: Suggesting friends using the implicit social graph. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)
47. Saito, K., Kimura, M., Ohara, K., Motoda, H.: Selecting Information Diffusion Models over Social Networks for Behavioral Analysis. In: Balcázar, J.L., Bonchi, F., Gionis, A., Sebag, M. (eds.) ECML PKDD 2010, Part III. LNCS, vol. 6323, pp. 180–195. Springer, Heidelberg (2010)
48. Shang, L., Yang, L., Wang, F., Chan, K., Hua, X.: Real-time large scale near-duplicate web video retrieval. In: ACM Multimedia 2010, pp. 531–540 (2010)
49. Song, C.: Mining and visualising wireless sensor network data Source. International Journal of Sensor Networks archive 2(5/6), 350–357 (2007)
50. Sozio, M., Gionis, A.: The community-search problem and how to plan a successful cocktail party. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)
51. Vaidya, J., Clifton, C.: Privacy preserving association rule mining in vertically partitioned data. In: Proc. of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2002)
52. Xie, L., Yan, R., Yang, J.: Multi-concept learning with large-scale multimedia lexicons. In: Proceedings of the International Conference on Image Processing, ICIP 2008, October 12-15, pp. 2148–2151. IEEE, San Diego (2008)
53. Yan, Y., Fung, G., Dy, J.G., Rosales, R.: Medical coding classification by leveraging inter-code relationships. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28. ACM (2010)
54. Yi, X., Zhang, Y.: Privacy-preserving distributed association rule mining via semi-trusted mixer. Data and Knowledge Engineering 63(2), 550–567 (2007)
55. Ying, Y., Campbell, C., Damoulas, T., Girolami, M.: Class Prediction from Disparate Biological Data Sources Using an Iterative Multi-kernel Algorithm. In: 4th IAPR International Conference on Pattern Recognition in Bioinformatics, Sheffield (2009)
56. Yu, H., Jianga, X., Vaidya, J.: Privacy-preserving SVM using nonlinear kernels on horizontally partitioned data. In: Proc. of the 2006 ACM Symposium on Applied Computing, Dijon, France, April 23-27 (2006)

57. Yan, X., He, B., Zhu, F., Han, J.: Top-K Aggregation Queries Over Large Networks. In: IEEE 26th International Conference on Data Engineering (ICDE), pp. 377–380 (2010)
58. Zhan, J., Matwin, S., Chang, L.: Privacy-preserving collaborative association rule mining. Journal of Network and Computer Applications 30(3), 1216–1227 (2007)
59. Zhang, C., Krishnamurthy, A., Wang, R.Y., Singh, J.P.: Combining Flexibility and Scalability in a Peer-to-Peer Publish/Subscribe System. In: Alonso, G. (ed.) Middleware 2005. LNCS, vol. 3790, pp. 102–123. Springer, Heidelberg (2005)
60. Zh. Zhao, J., Wang, H., Liu, J.: Ye, Yung Chang. Identifying biologically relevant genes via multiple heterogeneous data sources. In: Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27. ACM (2008)
61. Zhao, H., Lall, A., Ogihara, M., Jun, X.: Global iceberg detection over distributed data streams. In: Proc. of IEEE 26th International Conference on Data Engineering, ICDE (2010)
62. Zheng, L., Shen, C., Tang, L., Li, T., Luis, S., Chen, S., Hristidis, V.: Using data mining techniques to address critical information exchange needs in disaster affected public-private networks. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA (2010)
63. Zhu, Y., Fu, Y., Fu, H.: On privacy in time series data mining. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 479–493. Springer, Heidelberg (2008)
64. Wang, Y., Cong, G., Song, G., Xie, K.: Community-based greedy algorithm for mining top-K influential nodes in mobile social networks. In: Proc. of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA (2010)
65. White, B., Yeh, T., Lin, J., Davis, L.: Web-scale computer vision using MapReduce for multimedia data mining. In: Proceedings of the Tenth International Workshop on Multimedia Data Mining, MDMKDD 2010, ACM, New York (2010)
66. Wright, R., Yang, Z.: Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In: Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2004)
67. Wurst, M., Morik, K.: Distributed feature extraction in a p2p setting: a case study. Future Generation Computer Systems 23(1), 69–75 (2007)

Heterarchy in Constructing Decision Trees – Parallel ACDT

Urszula Boryczka, Jan Kozak, and Rafał Skinderowicz

Institute of Computer Science, University of Silesia, Będzińska 39,
41–200 Sosnowiec, Poland
{urszula.boryczka,jan.kozak,rafal.skinderowicz}@us.edu.pl

Abstract. In this paper, a novel decision tree construction algorithm that utilizes the Ant Colony Optimization (ACO) is presented. The ACO is a population based metaheuristic inspired by the foraging behavior of real ants. It consists in searching for optimal solutions by considering both local heuristic and accumulated (in the form of pheromone trails) knowledge.

In this paper we study a parallel version of the Ant Colony Decision Trees (ACDT) algorithm developed for constructing decision trees. Decision tree induction is a widely used technique to generate classifiers from training data through a process of recursively splitting the data attribute space. The main idea of this paper is to speed up the tree construction process by dividing the population of ants into subpopulations for which calculations are carried out in parallel. The exchange of information between ants is possible through direct and indirect communication channels on the local and global (inter-subpopulation) levels. Ants cooperating in this way form a structure called heterarchy.

A detailed study of the proposed algorithm, focusing both on the computation time and the quality of results, is carried out using data sets from the UCI Machine Learning repository. Proposed scheme of parallelization of the ACDT demonstrates the possibility to improve not only the computation time, but also the quality of results.

Keywords: Ant Colony Optimization, Ant Colony Decision Trees, heterarchy, parallel ACDT, decision tree.

1 Introduction

Ant Colony Optimization (ACO) is a metaheuristic approach to solve many different optimization problems by using principles of communicative behavior observed in real ant colonies. Ants can communicate with each other about paths they traversed by reinforcement mechanisms of pheromone trails laid on the appropriate edges. The pheromone trails can lead other ants to food sources. ACO was introduced in [16]. It is a population-based approach, where several generations of virtual ants search for good solutions. The following ants of the next generation are attracted by the pheromone changes so that they can search in the solution space near attractive solutions, concerning specific sub-spaces.

Ant-based approaches are good candidates for parallelization but not sufficient research has been done in the parallel version of ACO in the field of data mining so far. It seems quite easy how to parallelize them. Every processor can hold an ant or a colony of ants and after every generation/iteration number, the ants/colonies exchange information about their solutions. Most parallel ant colony algorithms follow this scheme and differ only in granularity and whether the computations for the updating pheromone information are done locally or globally by a master processor which distributes the new value of pheromone trails to the processors.

Recently some possible parallelization strategies for ACO have been proposed and classified into fine-grained and coarse-grained strategies [1]. In fine-grained parallelization strategies usually several artificial ants (or a simple ant) of a colony are assigned to each processor, therefore frequent information exchange between the small sub-colonies of ants (or between processors) takes place. Coarse-grained parallelization schemes run several colonies in parallel. This strategy is also referred to as a multi colony approach. The information exchange among colonies is done at certain intervals (number of iterations is established). Many parallel ACO applications to the combinatorial optimization problems have been analyzed [1, 9, 24–26, 32, 35, 36]. In [35], parallel MMAS with k independent runs was studied. The experiment performed using TSP instances showed superiority of parallel independent runs both in solution quality and computation time.

This article is organized as follows. Section 1 comprises an introduction to the subject of this article. In section 2, Ant Colony Optimization in Data Mining is presented. Section 3 describes decision trees and decision forests. Section 4 describes Ant Colony Decision Tree approach, especially the splitting rules. Section 5 focuses on the presented, new version of the parallel implementation of the ACDT approach. Section 6 presents the experimental study that has been conducted to evaluate the performance of the parallel ACDT, taking into consideration five data sets. Finally, we conclude with general remarks on this work and a few directions for future research are pointed out.

2 Ant Colony Optimization in Data Mining

Ant Colony Decision Trees (ACDT) algorithm [2, 4] employs ant colony optimization techniques [11, 17] for constructing decision trees and decision forests. Ant Colony Optimization is a branch of a newly developed form of artificial intelligence called swarm intelligence. Swarm intelligence is a form of emergent collective intelligence of groups of simple individuals: ants, termites or bees in which a form of indirect communication via pheromone was observed. Pheromone values encourage the ants following the path to build good solutions of the analyzed problem and the learning process occurring in this situation is called positive feedback or auto-catalysis.

In this paper we defined an ant algorithm to be a multi–agent system inspired by the observation of real ant colony behavior exploiting the stigmergic communication paradigm. The optimization algorithm in this paper was inspired by the previous works on Ant Systems (AS) and, in general, by the term — stigmergy. This phenomenon was first introduced by P. P. Grasse [22].

An essential step in this direction was the development of Ant System by Dorigo et al. [17], a new type of heuristic inspired by analogies to the foraging behavior of real ant colonies, which has proven to work successfully in a series of experimental studies. Diverse modifications of AS have been applied to many different types of discrete optimization problems and have produced very satisfactory results [14]. Recently, the approach has been extended by Dorigo et al. [12, 13, 18, 19] to a full discrete optimization metaheuristic, called the Ant Colony Optimization (ACO) metaheuristic.

The Ant Colony System (ACS) algorithm has been introduced by Dorigo and Gambardella to improve the performance of the Ant System [15, 17], which allowed to find good solutions within a reasonable time for small size problems only. The ACS is based on 3 modifications of the Ant System: a different node transition rule; a different pheromone trail updating rule; the use of local and global pheromone updating rules (to favor exploration).

An adaptation of Ant Colony Optimization to classification is a research area still not well explored and examined. The appeal of this approach similarly to the evolutionary techniques is that they provide an effective mechanism of conducting a more global search. These approaches are based on a collection of attribute–value terms, then it can be expected that these approaches will also cope better with attribute interaction than greedy induction algorithms [21].

The prototype of our approach Ant–Miner is an ant–based system [3, 30] and it is more flexible and robust than traditional approaches [31, 10]. The ACDT algorithm is a different approach than Ant–Miner. In the Ant–Miner algorithm the goal was to produce an ordered list of decision rules. The ACDT constructs decision trees.

3 Decision Trees and Decision Forests

A decision tree is used in determining the optimum course of action, in situations having several possible alternatives with uncertain outcomes. The resulting diagram displays the structure of a particular decision, and the interrelationships and interplay between different alternatives, decisions, and possible outcomes. Decision trees are commonly used in operational research, specifically in decision analysis, for identifying an optimal strategy for reaching a goal. The evaluation function for decision trees will be calculated according to the following formula:

$$Q(T) = \phi \cdot w(T) + \psi \cdot a(T, P) \tag{1}$$

where:

$w(T)$ – the size (numer of nodes) of the decision tree T,

$a(T, P)$ – the accuracy of the classification object from a test set P by the tree T,

ϕ and ψ – constants determining the relative importance of $w(T)$ and $a(T, P)$.

Constructing optimal binary decision trees is a NP–complete problem, where an optimal tree is one which minimizes the expected number of tests required for identification of the unknown objects (as shown by Hyafil and Rivest in 1976 [23]). The problem of designing storage efficient decision trees from decision tables was examined by Murphy and McCraw [27] in 1991. They showed that in most cases, the construction of the storage optimal decision tree is a NP-complete problem, and therefore a heuristic approach to the problem is necessary. Constructing an optimal decision tree may be defined as an optimization problem in which at each stage of creating decisions we select the optimal splitting of the data [33].

Classification And Regression Tree (CART) approach was developed by Breiman et al. in 1984 [7] and is characterized by the fact that it constructs binary trees. The splits are selected using the Twoing and Gini criteria. CART looks for splits that minimize the prediction squared error. A decision tree is built in accordance with a splitting rule that performs multiple splitting of a learning sample into smaller parts.

Twoing criterion will search for two classes that will make up together more then 50% of the data. Twoing splitting rule maximizes the following change-of-impurity measure which implies the following maximization problem for nodes m_l, m_r:

$$\underset{a_j \leq a_j^R, j=1,\ldots,M}{\arg\max} \left(\frac{P_l P_r}{4} \left[\sum_{k=1}^{K} |p(k|m_l) - p(k|m_r)| \right]^2 \right), \tag{2}$$

where:

$p(k|m_l)$ – the conditional probability of the class k provided in node m_l,
P_l – the probability of transition objects into the left node m_l,
P_r – the probability of transition objects into the right node m_r,
K – number of decision classes,
a_j – j–th variable,
a_j^R– the best splitting value of variable a_j.

A decision forest is a collection of decision trees. We defined the decision forest by following formula:

$$DF = \{d_j : X \rightarrow \{1, 2, ..., g\}\}_{j=1,2,...,J}, \tag{3}$$

where J, is a number of decision trees j ($J \geqslant 2$).

In decision forests, predictions of decision trees are combined to make the overall prediction for the forest. Classification is done by a simple voting. Each decision tree votes on the decision for the object and the decision with the highest number of votes is chosen. The classifier created by a decision forest DF, called the classifier $dDF : X \rightarrow 1, 2, ..., g$, uses the following voting rule:

$$dDF(x) := \arg\max_{k} N_k(x), \tag{4}$$

where:

k – decision class, such that $k \in \{1, 2, \ldots, g\}$,

$N_k(x)$ – number of votes for the object $x \in X$ classification in to class k, such that $N_k(x) := \#\{j : d_j(x) = k\}$.

In order to build a decision forest one needs a good method of constructing decision trees. Some approaches are described in [6, 8, 33].

4 Ant Colony Decision Trees Algorithm

Ant Colony Optimization (ACO) approach has been successfully applied to many difficult combinatorial problems. Ant Colony Decision Trees (ACDT) algorithm is the first ACO adaptation to the task of constructing decision trees. In order to improve the processing time of the Ant-Miner algorithm the parallel Ant-Miner algorithm was proposed and analyzed [5]. The classical algorithm proposed by Parpinelli et al. was not adapted to the continuous attributes, but some modifications or extensions to the Ant–Miner algorithm were proposed, for example the cAnt-Miner algorithm copes with continuous attributes during the rule construction process [28, 29, 34].

```
initialization_pheromone_trail(pheromone);
for $j = 1$ to number_of_iterations do
    best_tree := null;
    for $a = 1$ to ants_per_colony do
        new_tree := build_tree(pheromone);
        pruning(new_tree);
        assessment_of_the_quality_tree(new_tree);
        if new_tree is_of_higher_quality_than best_tree then
            best_tree := new_tree;
        endif
    endfor
    update_pheromone_trail(best_tree, pheromone);
endfor
result := best_constructed_tree;
```

Fig. 1. Pseudo-code of the ACDT algorithm

In each ACDT step an ant chooses an attribute and its value for splitting the objects in the current node of the constructed decision tree. The choice is made according to a heuristic function and pheromone values. The heuristic function is based on the Twoing criterion, which helps ants select an attribute-value pair which well divides the objects into two disjoint sets, i.e. with the intention that objects belonging to the same decision class should be put in the same subset. The best splitting is observed when similar number of objects is put into the left and right subtrees and objects belonging to the same decision class are in the same subtree. Pheromone values indicate the best way (connection) from the superior to the subordinate nodes – all possible combinations are taken into account.

The pseudo-code of the proposed algorithm is presented in Fig. 1. Lines 2–13 describe one iteration of this algorithm. Firstly, each ant builds a single decision tree (lines 4–11). Next, the best quality decision tree is selected and then the pheromone trail is updated on the edges used in the splits made during the construction of the tree. This process is repeated until the preset number of iterations is made.

As mentioned before, the value of the heuristic function is determined according to the splitting rule employed in CART approach (see formula (2)). The probability of choosing the appropriate split in the node is calculated according to a classical probability used in ACO:

$$p_{i,j} = \frac{\tau_{m,m_{L(i,j)}}(t)^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_i^a \sum_j^{b_i} \tau_{m,m_{L(i,j)}}(t)^{\alpha} \cdot \eta_{i,j}^{\beta}} \tag{5}$$

where:

$\eta_{i,j}$ – a heuristic value for the split using the attribute i and value j,

$\tau_{m,m_{L(i,j)}}$ – an amount of pheromone currently available at time t on the connection between nodes m and m_L, (it concerns the attribute i and value j),

α, β – the relative importance with experimentally determined values 1 and 3.

The initial value of the pheromone trail is determined similarly to the Ant–Miner approach and depends on the number of attribute values. The pheromone trail is updated (6) by increasing pheromone levels on the edges connecting each tree node with its parent node:

$$\tau_{m,m_L}(t+1) = (1-\gamma) \cdot \tau_{m,m_L}(t) + Q(T) \tag{6}$$

where $Q(T)$ is a quality of the decision tree (see formula (1)), and γ is a parameter representing the evaporation rate, equal to 0.1.

5 Parallel Implementation of ACDT Approach

The parallelization of the ant colony algorithms is difficult because of the necessity to maintain and frequently update a pheromone matrix. Especially, the local pheromone update rule requires a slight decrease of the pheromone level every time an ant selects an edge, which causes a strong functional dependency between successive iterations of the algorithm. Fortunately, in the ACDT algorithm the local pheromone update rule is not present and only the global pheromone update rule is used, which consists in depositing a pheromone trail on the edges belonging to the best solution (tree) found by ants. It opens a natural way to parallelize the ACDT algorithm by simply dividing the ants between multiple processors[1]. This strategy proved to be efficient in the case of the Ant–Miner algorithm for the induction of classification rules [5].

In the proposed Parallel-ACDT algorithm ants are evenly divided into subpopulations (colonies) and computations for the colonies are performed in parallel by all the available processors, thus the number of colonies is equal to the number of processors. Because the total number of ants in colonies is equal to the number of ants in the sequential ACDT algorithm, the total computational cost of the algorithm does not change[2].

```
ants_per_colony := number_of_ants/processors;
parfor $P_i, i = 0, 1, \ldots, processors - 1$ do
    initialize_pheromone_trail($pheromone_i$);
    $best\_tree_i$ := null;
    for $j = 1$ to number_of_iterations do
        for $a = 1$ to ants_per_colony do
            $new\_tree_a$ := build_tree($pheromone_i$);
            pruning($new\_tree_a$);
            assessment_of_the_quality_tree($new\_tree_a$);
            if $new\_tree_a$ is_of_higher_quality_than $best\_tree_i$ then
                $best\_tree_i$ := $new\_tree_a$;
            endif
        endfor
        { Exchange $best\_tree_i$ with other colonies (processors)
          according to an adopted cooperation strategy (cs) }
        $received\_tree_i$ := $exchange\_trees_{cs}$($best\_tree_i$);
        if $received\_tree_i$ is_of_higher_quality_than $best\_tree_i$ then
            $best\_tree_i$ := $received\_tree_i$;
        endif
        update_pheromone_trail($best\_tree_i$, $pheromone_i$);
    endfor
    Send $best\_tree_i$ to processor $P_0$; { $P_0$ is the master processor }
end
{ Produce $best\_tree_0$ as the result }
```

Fig. 2. Pseudo-code of the Parallel-ACDT algorithm (*cs* – chosen cooperation strategy – *all_to_all* or *heterarchic*)

As stated in [20] an *heterarchical* algorithm consists of agents which are not ranked, i.e. there is no hierarchy of importance between them and if they belong to different groups, there is no hierarchy of importance between the groups, too.

[1] We use the term *processor* referring to a single processing unit or core. Modern processors can contain multiple processing cores.

[2] Of course, the computational cost of the parallel algorithm is slightly larger because of the additional operations associated with the organization of parallel computations and the exchange of data between processors.

An important role in the heterarchical algorithms play *communication channels* between the agents or subsets of agents. In the Parallel-ACDT algorithm there are two communication channels present – *indirect* between the agents through pheromone trail and *direct* between the colonies through explicit solutions exchange. The latter is performed according to a preset *cooperation strategy.* There were two strategies investigated in the current work: an *all_to_all* strategy and a *heterarchic* strategy. Both will be described in details later in this section.

Pseudo-code for the parallel version of the ACDT algorithm is shown in Fig. 2. The main part of the algorithm is performed in parallel (lines 2–23). Each colony initializes and maintains its own pheromone matrix (line 3), which is updated once per iteration (line 20) after each ant in the colony finished building a decision tree. An essential part of the algorithm is the exchange of the iteration best tree (line 16) between the colonies, which is performed synchronously by the *exchange_trees* procedure. The *exchange_trees* procedure depends on the cooperation strategy used.

```
Procedure exchange_trees_all_to_all(local_best_tree)
  Input: local_best_tree – a tree to exchange with other colonies (processors)
  Result: The global best tree
1 rank := get_processor_rank();
2 Broadcast the quality of local_best_tree to other processors;
3 best_rank := Rank of the processor (colony) with the best quality tree;
4 if rank = best_rank then
5     Broadcast local_best_tree to other processors;
6     result := local_best_tree;
7 else
8     result := Receive a tree from the processor $P_{best\_rank}$;
9 end
```

Fig. 3. Pseudo-code of the exchange_trees procedure with the *all_to_all* cooperation strategy

The pseudo-code for the *exchange_trees* procedure with *all_to_all* strategy is shown in Fig. 3. The goal of the procedure is to select the global best solution (tree) from all the local best solutions found by the ants in all colonies. In order to reduce the amount of data send between processors only the quality of the local best tree is broadcasted (line 2) to other processors and afterwards only the processor with the highest quality tree sends it to the rest (line 5). Because all the processors have access to the global best tree, the execution of the Parallel-ACDT algorithm with the *all_to_all* strategy *is equivalent* to the execution of the sequential ACDT algorithm and the only result of performing computations in parallel can be a significant reduction of the algorithm's execution time. Although there is no hierarchy of importance between the agents of colonies in the Parallel-ACDT algorithm with *all_to_all* strategy, we may not refer to it as truly heterarchic, because all colonies are forced to have exactly the same

```
Procedure exchange_trees_heterarchic(local_best_tree)
  Input: local_best_tree – a tree to exchange with other colonys (processors)
  Result: A tree with higher quality selected between local_best_tree and a
          received one.
1 rank := get_processor_rank();
2 next := (rank + 1) mod processors;
3 prev := max(rank − 1, 0);
4 Send local_best_tree to the processor $P_{next}$;
5 received_tree := receive a tree from the processor $P_{prev}$;
6 if received_tree is_of_higher_quality_than local_best_tree then
7     result := received_tree;
8 else
9     result := local_best_tree;
10 end
```

Fig. 4. Pseudo-code of the exchange_trees procedure with the *heterarchic* cooperation strategy

pheromone trail values. As a result there is no notion of locality characteristic for the heterarchical algorithms [20].

The pseudo-code for the second exchange strategy, namely *heterarchic* strategy, is shown in Fig. 4. In this strategy all colonies are logically organized in a ring topology, i.e. a colony i, $i = 0, 1, \ldots p - 1$ (where p is the number of processors), sends its best tree to the colony $j, j = ((i + 1) \mod p)$, and receives a tree from the colony k, $k = max(0, i - 1)$, lines 4 and 5 respectively. It is worth mentioning, that a tree is send to the "next" colony before receiving a tree from the "previous" colony, therefore the global best tree is not available to all processors like in the *all_to_all* strategy. The consequence is that the execution of the Parallel-ACDT with the *all_to_all* cooperation strategy *is not equivalent* to the sequential execution of the ACDT algorithm. Because the global update of a pheromone trail is made using a colony's best tree or a best tree received from the previous colony, each colony may have a unique pheromone trail levels, i.e. pheromone matrix. In this way, the algorithm is similar to the multi colony ACS with two communication channels on the local and global levels. The first level involves indirect intra-colony information exchange (stigmergy) based on pheromone deposition. The second involves direct inter-colony information exchange in the form of the best solutions found.

6 Results of Experiments

The main objective of the experiments was to compare Parallel-ACDT algorithm with two cooperation strategies proposed, focusing both on the computation time and the quality of the results. For this purpose we selected 5 different tests from the UCI repository, namely: connect-4, krkopt, letter-recognition, pendigits and poker-hand. The following ACDT parameters were fixed: $q_0 = 0.0$, $\alpha =$

3.0, $\beta = 1.0$, *number_of_ants* $= 32$, *number_of_iterations* $= 10$. For each data set the Parallel-ACDT algorithm was run using a number of processors $p \in \{1, 2, 4, 8, 16, 32\}$ and the computations were repeated 30 times for each set of parameter values. Naturally, when $p = 1$ the execution of the Parallel-ACDT is equal to the execution of the sequential ACDT in which case there is only one non-cooperating ant colony.

The proposed parallel algorithms were implemented in the C++ programming language with Intel MPI Library v. 3.1. All computations were performed on a cluster of 336 nodes which consisted of 2 Intel Xeon Quad Core 2.33 GHz processors, each with 12MB level 3 cache. Nodes were connected with the Infiniband DDR fat-tree full-cbb network (throughput 20 Gbps, delay 5 μs). The computer was executing a Debian GNU/Linux operating system.

6.1 Classification Accuracy

In order to evaluate the quality of classifiers produced by the proposed algorithms each data set was divided into three disjoint samples: learning sample, test sample and validation sample. The first one was used to guide ants during the process of building a decision tree, the second one was used in order to prune the produced tree. The validation sample was not exposed to the algorithm during the learning process, it was used afterwards to evaluate the quality of the final classifiers obtained. Classification accuracy was measured as a ratio of the properly classified objects to the number of all objects. The ACDT algorithm produced both single tree and decision tree forest classifiers for each of the analyzed data sets. The former is just the best decision tree found and the latter consists of the iteration best trees found.

In Tab. 1 detailed classification accuracy rates are shown for the Parallel-ACDT algorithm with the two cooperation strategies proposed. As it can be seen, the quality of a single decision tree classifiers for the *heterarchic* cooperation strategy slightly decreases with the increasing number of colonies for all data sets, except *poker-hand*.

This effect can be explained if one considers how the Parallel-ACDT algorithm differs from the sequential ACDT algorithm. During a single iteration of the sequential ACDT algorithm a number of different trees is build by the ants, which are guided both by the heuristic knowledge and the pheromone trails deposited. Afterwards only the ant which built the best quality tree deposits pheromone trail. The more trees are built during an iteration of the algorithm, the higher quality of the best tree can be expected and, as a result, the search process becomes more focused on the high quality regions of the problem solution space. In the Parallel-ACDT algorithm the number of trees built by a single ant colony during a single iteration decreases when the number of colonies increases. It is not a problem in case of the *all_to_all* strategy, because the total number of trees built remains the same as in the sequential ACDT and the global best tree is sent to all colonies. On the other hand, it is a problem when the *heterarchic* strategy is used, because each colony has access to only its own best tree and the one received from other colony. As a result colonies have different pheromone

Table 1. Comparison of the classification accuracy for the decision trees and decision tree forests produced by the proposed Parallel-ACDT algorithms with two cooperation strategies (CR_{tree} – avg. classification rate for a single tree classifier, CR_{forest} – avg. classification rate for a forest classifier, δ_{forest} – std. deviation of CR_{forest})

Dataset	p	*All_to_all* strategy			*Heterarchic* strategy		
		CR_{tree}	CR_{forest}	δ_{forest}	CR_{tree}	CR_{forest}	δ_{forest}
connect-4	1	0.750	0.746	0.005	0.750	0.746	0.005
	2	0.752	0.747	0.004	0.751	0.780	0.005
	4	0.753	0.747	0.005	0.752	0.784	0.002
	8	0.751	0.746	0.006	0.751	0.786	0.001
	16	0.751	0.745	0.005	0.749	0.786	0.002
	32	0.752	0.748	0.005	0.747	0.787	0.002
krkopt	1	0.604	0.586	0.014	0.604	0.586	0.014
	2	0.605	0.591	0.012	0.605	0.690	0.006
	4	0.601	0.586	0.013	0.602	0.696	0.005
	8	0.604	0.588	0.011	0.596	0.697	0.006
	16	0.604	0.586	0.014	0.596	0.697	0.004
	32	0.602	0.587	0.014	0.586	0.697	0.006
letter-recognition	1	0.527	0.506	0.007	0.527	0.506	0.007
	2	0.523	0.506	0.009	0.524	0.565	0.006
	4	0.522	0.504	0.008	0.521	0.571	0.004
	8	0.522	0.506	0.007	0.517	0.574	0.005
	16	0.525	0.504	0.008	0.514	0.573	0.004
	32	0.521	0.508	0.010	0.512	0.575	0.005
pendigits	1	0.735	0.731	0.015	0.735	0.731	0.015
	2	0.738	0.731	0.014	0.736	0.795	0.009
	4	0.735	0.733	0.010	0.740	0.805	0.008
	8	0.732	0.732	0.014	0.738	0.808	0.006
	16	0.735	0.731	0.012	0.732	0.810	0.005
	32	0.740	0.734	0.018	0.721	0.816	0.004
poker-hand	1	0.516	0.519	0.015	0.516	0.519	0.015
	2	0.516	0.520	0.014	0.514	0.529	0.030
	4	0.519	0.518	0.014	0.518	0.568	0.014
	8	0.524	0.523	0.012	0.526	0.573	0.007
	16	0.518	0.521	0.014	0.528	0.572	0.007
	32	0.521	0.518	0.013	0.530	0.574	0.007

trails and the search process becomes more diverse. A positive effect of the search diversity in the *heterarchic* strategy can be seen if one compares the classification accuracy of the decision forest classifiers obtained for both cooperation strategies. For the *heterarchic* strategy, the classification accuracy was improved by more than 10% for the *krkopt* data set and 32 colonies (Tab. 1). Figure 5 shows that the classification accuracy of the decision forest classifiers raises greatly for 2 and further for 4 colonies (processors) and afterwards it stays on the similar level.

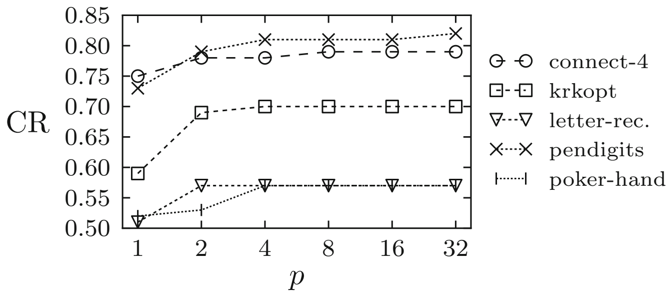


Fig. 5. Classification rate (CR) vs the number of colonies (p) for the Parallel-ACDT algorithm with the *heterarchic* cooperation strategy for the examined data sets

It appears that it could be possible to improve the quality of results obtained for the *all_to_all* strategy if not only the best quality tree, but several from the top of the ranking were sent to all processors. In this way the search process could be focused on a few (instead of a single) promising areas of problem solution space, but exchanging more trees will increase the communication overhead, which in turn may lower the speedup.

6.2 Performance Comparison

In Tab. 2 runtimes and speedups for the Parallel-ACDT algorithm with *all_to_all* and *heterarchic* cooperation strategies are presented for the following number of processors p ($p \in \{1, 2, 4, 8, 16, 32\}$). As it can be seen, both versions achieved similar runtimes and speedups in almost all cases. As it can be seen in Fig. 6 the algorithm with *heterarchic* cooperation strategy has a slight advantage over the algorithm with *all_to_all* strategy. For both strategies the speedups obtained were very good, almost linear, and in a few cases even superlinear, what is an

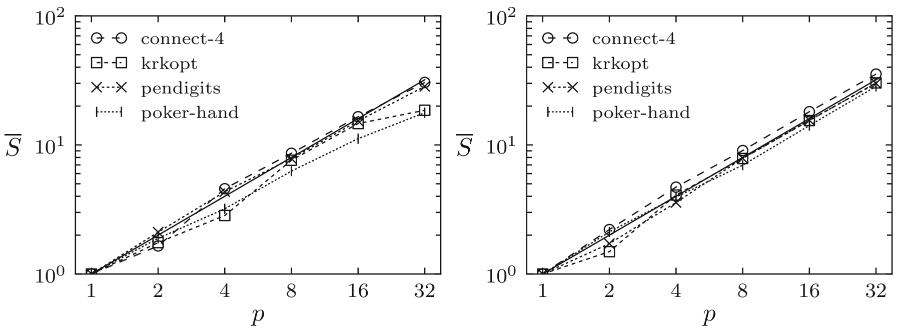


Fig. 6. Average speedups ($\overline{S}$) vs the number of processors (p) for the *all_to_all* (on the left) and the *heterarchic* (on the right) strategies. Speedups are shown in logarithmic scale. Solid line shows a linear speedup.

Table 2. Comparison of the parallel performance of the proposed Parallel-ACDT algorithm with two cooperation strategies (p – number of processors used, $\overline{t}$ – average runtime in seconds, δ – std. deviation of $\overline{t}$, $\overline{S}$ – average speedup)

Dataset	p	*All_to_all* strategy			*Heterarchic* strategy		
		$\overline{t}$	δ	$\overline{S}$	$\overline{t}$	δ	$\overline{S}$
connect-4	1	197.85	8.65	1.00	197.85	8.65	1.00
	2	119.63	1.60	1.65	89.70	14.43	2.21
	4	43.24	0.76	4.58	41.99	0.73	4.71
	8	22.95	0.24	8.62	21.87	0.18	9.05
	16	11.96	0.18	16.54	10.94	0.11	18.09
	32	6.49	0.17	30.49	5.59	0.05	35.39
krkopt	1	36.01	1.70	1.00	36.01	1.70	1.00
	2	20.55	4.32	1.75	24.14	9.82	1.49
	4	12.68	4.94	2.84	8.81	0.05	4.09
	8	4.73	0.28	7.61	4.59	0.03	7.85
	16	2.46	0.04	14.64	2.33	0.01	15.45
	32	1.94	0.58	18.56	1.19	0.04	30.26
letter-recognition	1	148.49	5.95	1.00	148.49	5.95	1.00
	2	69.48	0.40	2.14	95.00	31.93	1.56
	4	36.23	0.19	4.10	45.66	16.43	3.25
	8	17.26	0.10	8.60	17.02	0.11	8.72
	16	8.80	0.24	16.87	8.34	0.06	17.80
	32	5.31	0.25	27.96	4.17	0.10	35.61
pendigits	1	222.49	6.41	1.00	222.49	6.41	1.00
	2	105.31	3.25	2.11	129.58	42.08	1.72
	4	51.32	2.62	4.34	62.22	17.88	3.58
	8	28.39	1.10	7.84	28.24	0.41	7.88
	16	14.50	0.30	15.34	14.32	0.17	15.54
	32	7.83	0.26	28.42	7.42	0.12	29.99
poker-hand	1	194.69	14.10	1.00	194.69	14.10	1.00
	2	103.00	6.60	1.89	91.66	6.35	2.12
	4	60.81	5.74	3.20	49.29	0.88	3.95
	8	31.06	1.67	6.27	27.93	0.25	6.97
	16	17.41	0.68	11.18	13.78	0.08	14.13
	32	10.95	0.48	17.78	6.85	0.07	28.42

effect of the increased size of the accumulated (processors) cache memory. For two data sets, namely *krkopt* and *poker-hand* the speedups obtained for the Parallel-ACDT with the *all_to_all* strategy were significantly worse than for the *heterarchic* strategy, especially for the 32 processors. It is a result of the increased inter-processor communication overhead required by the first strategy in which *all colonies* exchange informations about the best trees found so far. Indeed, in order to find the global best tree in the *all_to_all* strategy, there are $2p \log p$

inter-processor send-receive operations needed compared to p for the *heterarchic* strategy, where p is the number of processors (colonies) used. Moreover, the exchange of information for the first strategy requires also a synchronization of all processors, whereas for the latter strategy such synchronization is required only for the pair of processors exchanging solutions between themselves (Fig. 4). The slopes of the speedup plots (Fig. 6), which are close to the slope of the linear speedup (solid line), suggest that both algorithms should scale well for the numbers of processors far beyond 32.

7 Conclusions

A parallel version of the ACDT algorithm for constructing decision trees and decision forests was presented in which several ant colonies cooperate periodically. Ants in the colonies cooperate on the two levels: intra-colony using a pheromone trail and inter-colony using explicit information exchange in the form of the best quality solutions found. Two inter-colony cooperation strategies were proposed and tested for classification accuracy and computation performance on the selected data sets from well-known UCI repository. For both strategies the computation times and speedups obtained were very good up to 32 processors, with slight advantage of the *heterarchic* strategy. This strategy also proved to be the best in terms of classification accuracy of the decision forest classifiers, which suggests that the diversity of the decision trees is very important, even if individual trees are not of the best quality. Of course, one may tray to use more elastic cooperation strategies, than those proposed in this work. It may result in further improvement in the quality of the individual decision trees and the decision forest classifiers produced.

Acknowledgements. We thank the Academic Computer Centre in Gdansk TASK where the computations of our project were carried out.

References

1. Benkner, S., Doerner, K., Hartl, R., Kiechle, G., Lucka, M.: Communication strategies for parallel cooperative ant colony optimization on clusters and grids. In: Complimentary Proc. of PARA 2004 Workshop on State-of-the-art in Scientific Computing, pp. 3–12 (2005)
2. Boryczka, U., Kozak, J.: Ant colony decision trees – A new method for constructing decision trees based on ant colony optimization. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) ICCCI 2010, Part I. LNCS, vol. 6421, pp. 373–382. Springer, Heidelberg (2010)
3. Boryczka, U., Kozak, J.: New Algorithms for Generation Decision Trees – Ant–Miner and Its Modifications. In: Abraham, A., Hassanien, A.-E., de Leon F. de Carvalho, A.P., Snášel, V. (eds.) Foundations of Computational Intelligence 6. SCI, vol. 206, pp. 229–264. Springer, Heidelberg (2009)
4. Boryczka, U., Kozak, J.: An adaptive discretization in the ACDT algorithm for continuous attributes. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) ICCCI 2011, Part II. LNCS, vol. 6923, pp. 475–484. Springer, Heidelberg (2011)

5. Boryczka, U., Kozak, J., Skinderowicz, R.: Parellel Ant–Miner. Parellel implementation of an ACO techniques to discover classification rules with OpenMP. In: 15th International Conference on Soft Computing, MENDEL 2009, pp. 197–205. University of Technology, Brno (2009)
6. Breiman, L.: Random forests. Mach. Learn. 45, 5–32 (2001)
7. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Chapman & Hall, New York (1984)
8. Bühlmann, P., Hothorn, T.: Boosting algorithms: Regularization, prediction and model fitting. Statistical Science 22(4), 477–505 (2007)
9. Bullnheimer, B., Kotsis, G., Strauss, C.: Parallelization strategies for the ant system. In: High Performance Algorithms and Software in Nonlinear Optimization, pp. 87–100 (1998)
10. Clark, P., Niblett, T.: The CN2 rule induction algorithm. Machine Learning 3(4), 261–283 (1989)
11. Corne, D., Dorigo, M., Glover, F.: New Ideas in Optimization. Mc Graw–Hill, Cambridge (1999)
12. Doerner, K.F., Merkle, D., Stützle, T.: Special issue on ant colony optimization. Swarm Intelligence 3(1), 1–2 (2009)
13. Dorigo, M., Caro, G.D.: New Ideas in Optimization. McGraw-Hill, London (1999)
14. Dorigo, M., Caro, G.D., Gambardella, L.: Ant algorithms for distributed discrete optimization. Artif. Life 5(2), 137–172 (1999)
15. Dorigo, M., Gambardella, L.M.: Ant Colony System: A cooperative learning approach to the Traveling Salesman Problem. IEEE Trans. Evol. Comp. 1, 53–66 (1997)
16. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: an autocatalytic optimization process. Tech. Rep. 91-016, Department of Electronics, Politecnico di Milano, Italy (1996)
17. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
18. Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.): ANTS 2008. LNCS, vol. 5217. Springer, Heidelberg (2008)
19. Dorigo, M., Birattari, M., Stützle, T., Libre, U., Bruxelles, D., Roosevelt, A.F.D.: Ant colony optimization – artificial ants as a computational intelligence technique. IEEE Comput. Intell. Mag. 1, 28–39 (2006)
20. Dréo, J., Siarry, P.: Continuous interacting ant colony algorithm based on dense heterarchy. Future Generation Computer Systems, pp. 841–856 (2004)
21. Galea, M.: Applying swarm intelligence to rule induction. Master's thesis, University of Edingbourgh (2002)
22. Grasse, P.P.: Termitologia, Paris, Masson, vol. II (1984)
23. Hyafil, L., Rivest, R.: Constructing optimal binary decision trees is NP–complete. Inf. Process. Lett. 5(1), 15–17 (1976)
24. Lv, Q., Xia, X., Qian, P.: A parallel ACO approach based on one pheromone matrix. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 332–339. Springer, Heidelberg (2006)
25. Manfrin, M., Birattari, M., Stützle, T., Dorigo, M.: Parallel ant colony optimization for the traveling salesman problem. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 224–234. Springer, Heidelberg (2006)
26. Middendorf, M., Reischle, F., Schmeck, H.: Multi colony ant algorithms. J. Heuristics 8(3), 305–320 (2002)

27. Murphy, O., McCraw, R.: Designing Storage Efficient Decision Trees. IEEE Transactions on Computers 40, 315–320 (1991)
28. Otero, F.E.B., Freitas, A.A., Johnson, C.G.: *c*Ant-miner: An ant colony classification algorithm to cope with continuous attributes. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) ANTS 2008. LNCS, vol. 5217, pp. 48–59. Springer, Heidelberg (2008)
29. Otero, F.E.B., Freitas, A.A., Johnson, C.G.: Handling continuous attributes in ant colony classification algorithms. In: CIDM, pp. 225–231 (2009)
30. Parpinelli, R.S., Lopes, H.S., Freitas, A.A.: Data mining with an ant colony optimization algorithm. IEEE Transactions on Evolutionary Computation, Special issue on Ant Colony Algorithms, 321–332 (2004)
31. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
32. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. J. Parallel Distrib. Comput. 62(9), 1421–1432 (2002)
33. Rokach, L., Maimon, O.: Data Mining With Decision Trees: Theory And Applications. World Scientific Publishing (2008)
34. Schaefer, G.: Ant colony optimisation classification for gene expression data analysis. In: Sakai, H., Chakraborty, M.K., Hassanien, A.E., Ślęzak, D., Zhu, W. (eds.) RSFDGrC 2009. LNCS, vol. 5908, pp. 463–469. Springer, Heidelberg (2009)
35. Stützle, T.: Parallelization strategies for ant colony optimization. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 722–731. Springer, Heidelberg (1998)
36. Talbi, E.G., Roux, O.H., Fonlupt, C., Robillard, D.: Parallel ant colonies for the quadratic assignment problem. Future Generation Comp. Syst. 17(4), 441–449 (2001)

Decision Support Simulation System Based on Synchronous Manufacturing

F. Javier Otamendi

Universidad Rey Juan Carlos, Campus Vicálvaro
Facultad de Ciencias Jurídicas y Sociales
Departamento Economía Aplicada I
Paseo Artilleros s/n
28032 Madrid, Spain
franciscojavier.otamendi@urjc.es

Abstract. The focus of the article was the redesign of an assembly cell in a car manufacturing company. The new design had to include the operations related to the assembly of new roofs without disrupting the throughput rate. The theory-of-constrains framework has been used to propose designs as well as to define the evaluation criteria. The complexity of the system has been studied using a discrete-event simulation model. The traditional scholar steps have been followed to develop a robust decision support simulation system (DSSS).

Keywords: Decision support systems, simulation, automotive systems.

1 Introduction

The decision problems faced in industry, commerce, public administration, and the society in general keep growing in size and complexity. For the study of these decision problems, it is necessary to develop efficient methodologies and tools, so it is possible to try and evaluate many different alternatives and to take the correct complex decision in a reasonable amount of time.

This is the case of a company that assembles vehicles, which is faced with the decision making problem of redesigning one of the main lines within the plant, more specifically, the cell that is located before the pre-glazing station. Due to a change in the design of the body of the vehicle and the possibility of including different types of roofs, the layout of the cell needs to be changed. Some bodies will now incorporate a new part, that needs to be either manually or automatically assembled depending on the vehicle model. The objective of the redesign is that the cycle time at the entry of the pre-glazing station must be maintained for the throughput rate to be same.

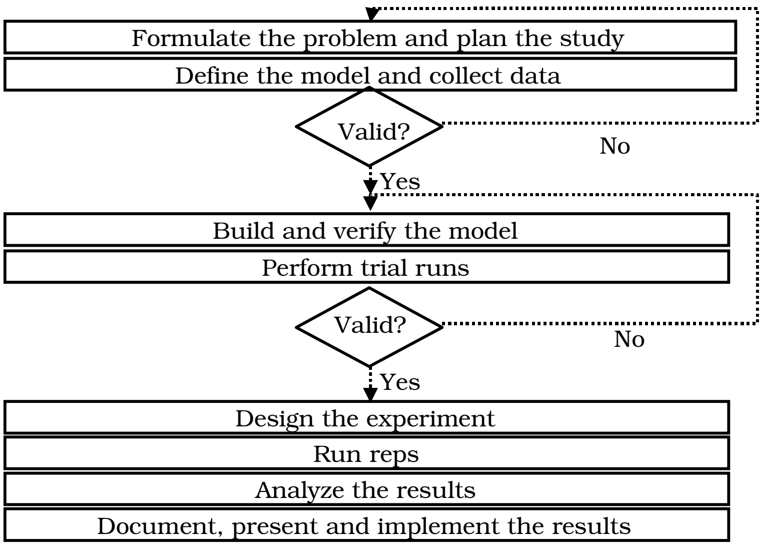
However, this new assembly mix puts too much stress in the assembly cell reaching the point where the throughput rate could be reduced considerably. Different layouts should be generated and evaluated, and the best one selected in terms of many criteria, like throughput and work-in-progress.

The complexity of the study comes due to the following reasons:

- The complexity of the system: the logic that governs the relationship between the stations and the demand mix is complex.
- The stochastic nature of any real system: the criteria or objective functions are not deterministic since the input data is random.
- The multicriteria nature of the problem: the objective function must be an aggregation of several individual functions.
- The large number of alternative layouts that might be defined.

Fortunately, some of the complexity of these studies has been diminished by the improvement not only of the solution techniques but also of the information technology. These improvements call these days for the experimentation with models instead of with the real system. Among these models, simulation has grown as one of the most reliable abstraction tools due to its very good compromise between the level of detail in the representation of the complex real system and the execution time of the model, which calls for an appropriate experimentation and decision making. The outcome therefore needs to be a special class of decision support system (DSS), called decision support simulation system (DSSS) [12].

To perform a sound simulation modeling study based on a DSS, the phases that are included in Figure 1 [6] must be followed. The first step is to define the problem and state the objectives. Once the situation is completely understood both by the management of the premises and the simulation expert, a simulation model might be developed. Some runs must then be tried to verify the correct execution of the model. The validity of the model must also be assessed in terms both of quantification of objectives and execution time. Then, an easy-to-use interface might be developed to facilitate the experimentation phase by easily keying data in and reading the results out. The combination of model and interface is a powerful DSSS. Then, different alternatives might be tried by changing input values and the best alternative selected.



Formulate the problem and plan the study
Define the model and collect data
Valid?
No
Yes
Build and verify the model
Perform trial runs
Valid?
No
Yes
Design the experiment
Run reps
Analyze the results
Document, present and implement the results

Fig. 1. Phases of a Simulation Study

What should become clear from this short description is that this modeling approach to decision making has one big part of science, with sound programming, logic and mathematical background, but it also has another part of art, with lots of imagination and intuition.

What follows is the summary of the simulation study that was performed in the automotive assembly plant. The presentation follows the phases just presented, with emphasis on the always important artistic side of the study. Section 2 includes a description of system, which includes the definition of the decision criteria and presentation of the available data. Section 3 is then used to describe the model development phase and Section 4 the results after simulating the three alternative designs that are going to be screened. Section 5 is then used to perform the comparison both quantitatively and qualitatively. Section 6 concludes.

1.1 Description of the System

The first step of a simulation study is the specification of the system in terms of the layout of the facilities, the flows of the units, and the available data, with the necessary declaration and quantification of the objectives that must be achieved.

1.1.1 Current Layout

The layout of the current assembly process is depicted in Figure 2. After the bodies have gone through a series of operations, the rolling conveyor system moves the bodies to the buffer of the assembly cell. The input buffer is composed of a line buffer that may hold up to five units before it feeds three parallel lines of conveyors, in which several check operations are performed.

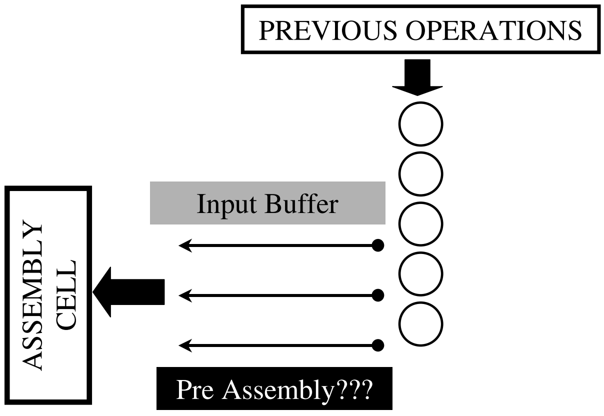


Fig. 2. Current layout

There exists then a transfer system that moves the bodies into the assembly cell. The selection of the next body is controlled by a powerful information system which is in charge of maintaining and defining the proper sequence into the assembly cell.

1.1.2 Available Data

Demand Mix

The first and most important piece of information is that of the demand. The reason for the study is the new product mix that has resulted from the possibility of including an additional part in the vehicle. This extra part might considerably vary the demand mix and therefore the capacity potential of the assembly and the pre-assembly cell. The demand in this case is not known but forecasted, and in fact, the reason for the study is to assess the demand (capacity) that each design might hold. On this screening phase, the total demand is maintained as its current level of 1 every 60 seconds (input and output cycle time) but with the mix included in Table 1 according to the new parts and resources that are needed to perform the pre-assembly.

Table 1. Demand Mix

TYPE	PERCENTAGE
MANUAL	9.0%
AUTOMATIC	29.5%
BOTH	3.8%
NONE	57.7%

The second set of data is that of the sequence. Due to technology restriction, only one manual operation might be performed every three bodies. Also, and due to set-ups only two robot operations could be performed on a row. Combining all the restrictions, a repeated sequence of 30 bodies has been used as a base sequence to perform the comparisons.

Times and Distances

The layout of the plant is provided so the distances are easily measured, including those of the conveyors, the transfers and the buffers. The speeds of the conveyor under normal operations are also specified. Set-up and assembly times are available in the process specifications.

Breakdowns, Maintenance and Repair Times

Breakdowns and maintenance data is also available as might be critical [7]. The possibility of disturbances is incorporated using the factory historic data. The mean-times-before-failure (MTBF) as well as the mean-times-to-repair (MTTR) are statistically provided in the form of random variables.

1.2 Planning the Study

The new product design has forced a redesign of the process, and that includes the necessity to perform pre-assembly operations in the buffer area, either manually, automatically or both. In fact, the aim of this study is to expand and redefine the waiting area. The reengineering task is the main topic covered in this article and includes the definition of different alternative layouts, their comparison using simulation and the selection in terms of the synchronization of cycle times as well as the minimization of the investment costs.

Synchronous manufacturing (also known as Optimized Process Technology – OPT, Theory of Constraints – TOC or Drum-Buffer-Rope) is a manufacturing philosophy that aims at balancing the flows of each of the stages of the process and not their capacity (Goldratt 2004). Using simulation to understand synchronous systems is a current line of research [2, 8, 10].

1.3 Objectives

Setting the objectives is too crucial to take it lightly. The objectives not only give a numerical quantity to test and compare the different alternatives, but also may have an influence in the modeling. The level of detail in the model should be enough to be able to calculate an accurate value for different objectives, but not too deep to slow the execution time of the simulation runs. The proper compromise must be achieved to successfully perform the study.

Setting the aspiration levels and the satisficing thresholds properly will determine the degree of fulfillment of the requirements as well as the best of the available alternatives. In this case, the requirements are set following the synchronous manufacturing philosophy [1]:

- “The process must keep the cycle time“ or “the throughput rate must be maintained or improved”
- “The work-in-progress should be minimized”

Summarizing, the main objective is to guarantee the current cycle time both at the entrance and the exit of the pre-assembly cell, which should coincide with the cycle time of the assembly cell [4].

2 Model Development: Decision Support Simulation System

Once the system is completely defined and the objectives understood and quantitatively specified, the important time to develop the model and the interface that combine into a DSSS arrives. This step has usually more of an art than a science. The developer has first to be expert enough to use a software package that fulfills the modeling needs as well as one that is familiar enough to reduce the developing time. And secondly, the level of detail of the model must be set in order to correctly estimate the value for the objectives within a reasonable development and execution time. The first idea usually is to go to full detail [3] but it is very important to just concentrate on the estimation of the criteria.

2.1 Selection of the Modeling Tool

Any complex manufacturing problem (and by extension any supply chain) might be simulated by using one of four different approaches: spreadsheet simulation, system dynamics, discrete-event dynamics systems and business games [5]. To develop the decision support system, the call was to develop the simulation model using discrete-event methods and tools, and using spreadsheets for easy input and output of data.

A complete survey of simulation software is performed biannually by Swain and published by OR/MS Today [9]. Forty-eight products are listed in the operations research and management science field. Each software tool is described in terms of typical application and market orientation, system requirements, model building, animation, support/training, pricing and vendor information, and new major features. In this case, the model was performed in Witness as it was the preferred one by the company. Besides, it is the tool that the research team is most familiar with.

2.2 The Model Logic

Modelling in Witness is performed in three steps. First, the objects are drawn out of a standard library. Then, each object is particularized, including name, graphical appearance and personal characteristics (capacity, process times, and others). Finally, the logic of movement is included. Although easy to develop and maintain, the interface in 2D is usually not appealing. The software has however the possibility to 207 develop a 3D interface, but this development is usually too costly and time consuming. The MsExcel interface allows to interchange input or output data easily and to launch the simulation while running the software on the background.

The model is built with the traditional combination of parts (bodies), buffers for the parts to wait and machines to perform the operations. The modelling philosophy is based on a PULL strategy following synchronous manufacturing principles. The machine representing the pre-glazing station pulls from the buffer that is at its entrance. This buffer is fed by a machine that represents the transfer from the lines. This transfer is a key element, since it selects from the different lines the next finished body that goes into the pre-glazing station according to the sequence number. This number has been implemented with an attribute so as to accompany the part throughout the execution of the model. Each body gets its sequence number at the entry buffer to the whole cell.

This entry buffer is critical since it assigns all the information about the sequence and the characteristics of body type and roof to the part. Machine times will be assigned accordingly. This buffer is emptied with a machine that directs the parts according to their attribute to the proper line. Then, a series of machines represent the conveyors, moving the bodies in between machines. The travel times at these machines correspond to the movement time of the machine, except for the machines that represent the manual or the robot operation, which will then last as needed.

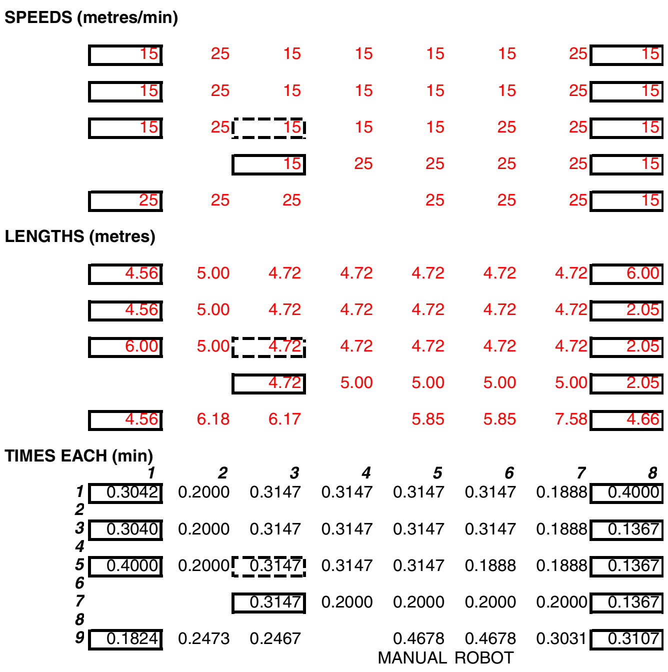
The simulation is therefore straight forward except for the entry and the exit, as well as the parameterisation of the elements in terms of the input data and the specifications of the results, all of which are either read from or output to the MsExcel interface.

2.3 Interface

Visualization and experimentation are the forte of simulation tools, although the commercial tools available are not usually very user friendly. In fact, most of the available commercial packages have incorporated and rely heavily on their MsExcel connectivity to input data and to show results. The current trend in the market is to develop the interface in spreadsheets or databases, which shows a button to execute the simulation and presents the main results in an environment that is familiar to the user. The simulation is performed in the background without any interaction or visualization.

2.3.1 Input Data

The required data is included in a worksheet of an MsExcel file. Figure 3 includes the layout data as well as the movement times at each machine. Several other input sheets are available in the interface for demand data but not included here as they are susceptible pieces of information like the demand and the sequence mix.



SPEEDS (metres/min)

15	25	15	15	15	15	25	15
15	25	15	15	15	15	25	15
15	25	15	15	15	25	25	15
		15	25	25	25	25	15
25	25	25		25	25	25	15

LENGTHS (metres)

4.56	5.00	4.72	4.72	4.72	4.72	4.72	6.00
4.56	5.00	4.72	4.72	4.72	4.72	4.72	2.05
6.00	5.00	4.72	4.72	4.72	4.72	4.72	2.05
		4.72	5.00	5.00	5.00	5.00	2.05
4.56	6.18	6.17		5.85	5.85	7.58	4.66

TIMES EACH (min)

	1	2	3	4	5	6	7	8
1	0.3042	0.2000	0.3147	0.3147	0.3147	0.3147	0.1888	0.4000
2								
3	0.3040	0.2000	0.3147	0.3147	0.3147	0.3147	0.1888	0.1367
4								
5	0.4000	0.2000	0.3147	0.3147	0.3147	0.1888	0.1888	0.1367
6								
7			0.3147	0.2000	0.2000	0.2000	0.2000	0.1367
8								
9	0.1824	0.2473	0.2467		0.4678	0.4678	0.3031	0.3107
					MANUAL	ROBOT		

Fig. 3. Screenshot of the input page

2.3.2 Output Criteria

Similarly, the interface will include the values of the output variables. The alternative designs are going to be evaluated in detailed according to the following criteria:

- Percent Blockage at entrance
- Occupation of "Entry Transfer"
- Occupation of "Automatic" operation
- Occupation of "Manual" operation
- Occupation of "Output Transfer"
- Average and Maximum Number at Input Buffer before Entry Transfer
- Average and Maximum Number of Bodies in New Area
- Average Cycle Time
- Possible Flow Improvement = Idle Percentage (Starving) for Entry to Assembly

This thorough list of output variables relates to the evaluation criteria of synchronous manufacturing (TOC). Under this philosophy, the cycle time has to correspond with the demand and it should provoke a tense, lean flow at each and every stage of the process. Then, the occupation of the all the processes should be balanced at about 90% of utilization, including set-ups and downtimes [11].

Figure 4 depicts the screen that has been developed to show a possible design cell with the old and new transfer lines, including the direction of flow, as well as the main results: average bodies or work-in-progress (WIP), capacity or maximum instantaneous WIP and the occupation ratio. Figure 5 then shows the output table as well as the pie chart corresponding to the occupation of any of the stations. Figure 6 finally shows the part of the output interface with the most important criteria: the cycle time.

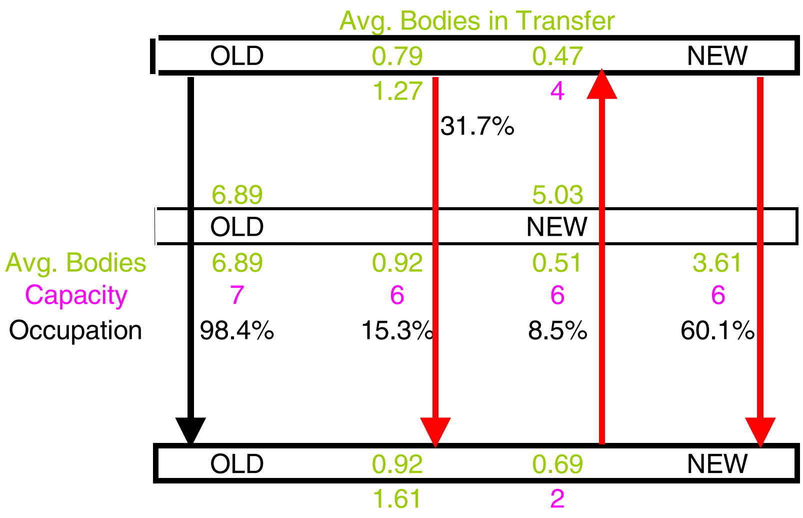
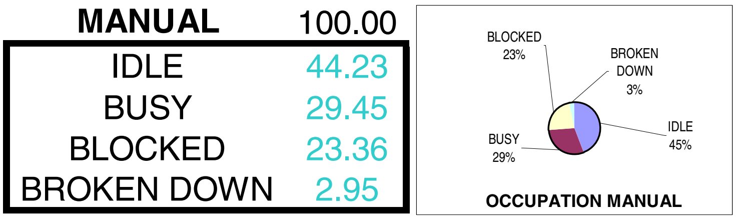
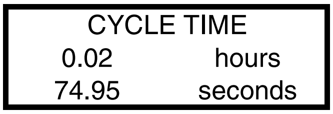


Fig. 4. Screenshot of the results in MsExcel (i)



MANUAL	100.00
IDLE	44.23
BUSY	29.45
BLOCKED	23.36
BROKEN DOWN	2.95

Fig. 5. Screenshot of the results in MsExcel (ii)



CYCLE TIME	
0.02	hours
74.95	seconds

Fig. 6. Screenshot of the results in MsExcel (iii)

2.4 Verification and Validation

Several trial runs were iteratively performed to fulfill the requisites of both the simulation experts and the management of the premises. The models behaved as intended so the models was readily verified. Then, the results with the current layout were presented to the management so they were confident that the model was running correctly and that the DSSS was ready to start with the evaluation, comparison and selection of the alternative designs.

3 Simulation Experiments

The decision support simulation system (DSSS) has been developed to evaluate the different layout designs and to select one for implementation. Many alternative designs were proposed but a first screen based on physical limitations as well as required investments brought the number down to just three options. What follows is the definition and evaluation of the three options, namely, A, B and C.

3.1 Layout Options

3.1.1 Option A (3+1)

The first possibility is to provide room for an additional line in parallel to the current three (Figure 7). The new part will be assembled in the new line, so the bodies that need to incorporate the part will take this pre-assembly line composed of two stations: one manual and the other automatic. The rest of the bodies will use the original lines. In the new line some bodies will suffer a delay due to the lack of capacity.

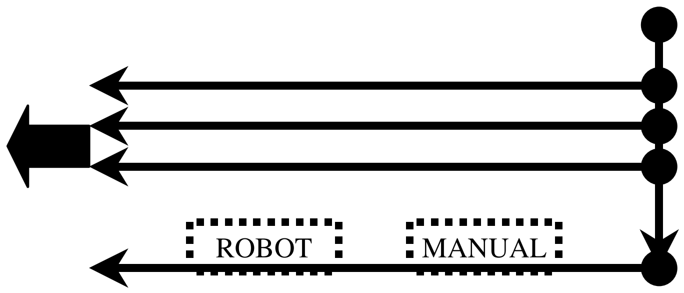


Fig. 7. Layout for Option A

Of course, this layout releases stress out of the old lines, that have now less bodies to move. The quantitative study has to assess if the new line is able to cope with the demand for the new bodies. A capacity problem might appear also at the transfer line that is now four lines wide with the corresponding increase in movement and therefore in transfer times. The input throughput rate to the buffer system might also be in jeopardy if the new line is not able to cope with the new tasks, blocking the entrance and not allowing for the required cycle time at the assembly cell.

3.1.2 Option B

To provide flexibility and minimize the blocking probability at the entrance, the second option looks at the possibility of using the third old line also as an input buffer to the new line. This layout (Figure 8) implies that the third line moves the bodies in the opposite direction to the flow. If the new line is full, the bodies that require pre-assembly will take the second line and then the third before they reach the new buffer line.

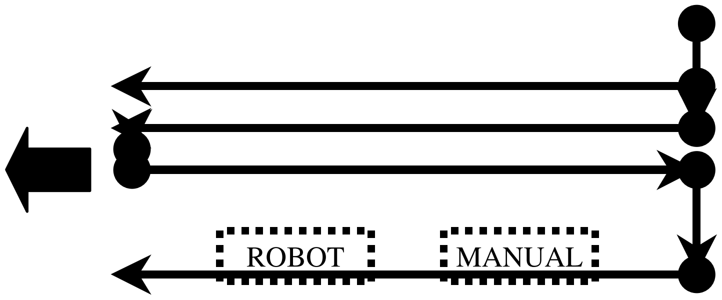


Fig. 8. Layout for Option B

The stress is now put on the bodies that will not go through the new operation. The feasibility of this design clearly depends on the production mix, as always, but even more so in this case.

3.1.3 Option C

The last option is to differentiate between the two types of bodies by including an additional line in which the manual assembly of parts could be performed, leaving the automatic pre-assembly for the last line (Figure 9).

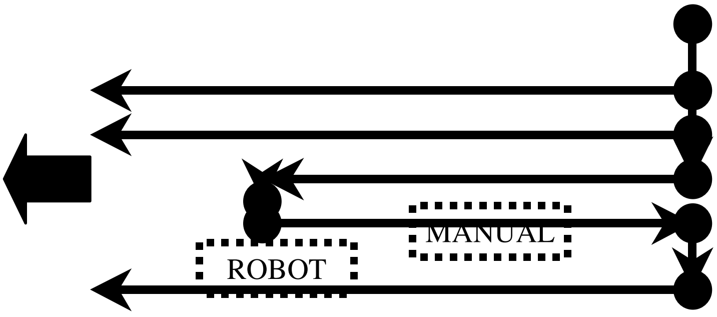


Fig. 9. Layout for Option C

This extra investment with a higher cost will a priori allow for a higher throughput with more flexibility and less blockage at the entrance.

4 Results of the Simulation Runs

The results for each of the simulated options are shown in graphical form in Figure 10, Figure 11 and Figure 12, for Options A, B, and C respectively. The graphs correspond to the MsExcel interface that was built to easily transfer data to and from the simulator.

Option A shows, as foreseen a very high occupation rate in the new line (97.9%), which provokes a high blockage percentage at the entrance (42.76%) as well as in the manual (62.49%) and the robot operations (77.57%). The output rate (cycle time) however is enough to keep the pace of the assembly cell (66.50 seconds), which is the one that cannot cope with more work (0.21% idle time) due to its high downtime (9.57%).

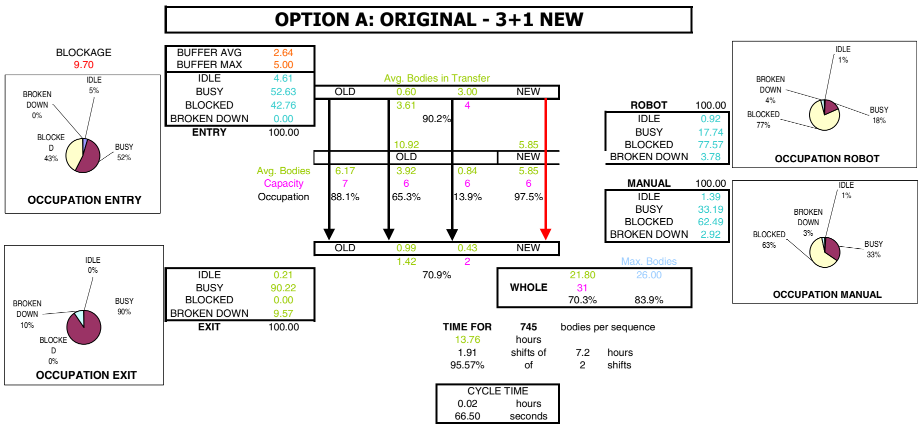


Fig. 10. Results for Option A

Option B puts too much stress on the old system (above 98% occupation), blocking the entrance 53.72% of the time. The input cycle time dictates then the output cycle time (74.95 seconds), which is too high allowing for idle time in the assembly cell (11.05%), the manual operation (44.23%) and the robot (66.88%).

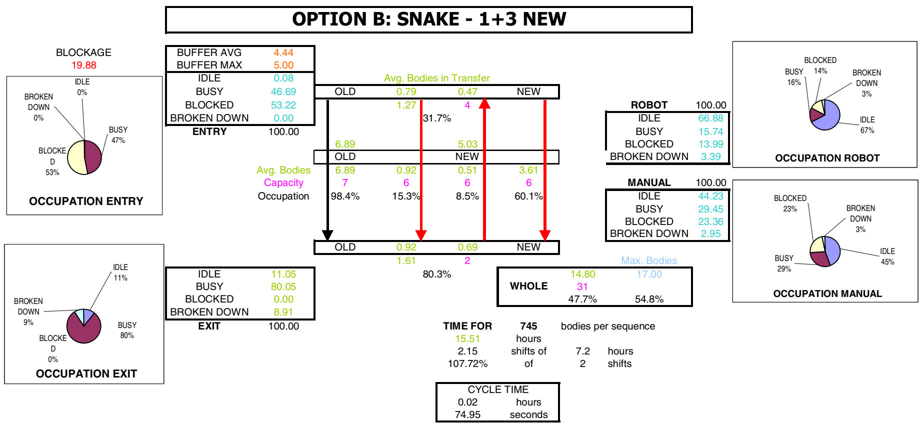


Fig. 11. Results for Option B

Option C is more balanced with cycle times similar to those of the assembly cell (67.09 seconds). There are no idle times in any station, although the occupation rates of the old lines are too high (98%).

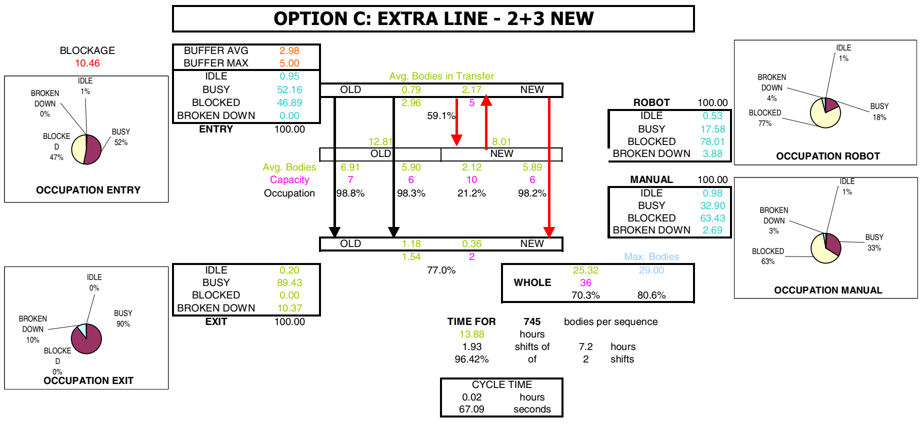


Fig. 12. Results for Option C

5 Comparison and Selection

Table 2 summarizes the behaviour of the three designs in quantitative terms, choosing the main criteria. Buffer space refers to the maximum designed capacity whereas average and maximum bodies correspond to the dynamic behaviour. The cycle time corresponds to the throughput potential and the improvement to the blockage ratio, which could be eliminated through investments or further process improvement.

Table 2. Comparison of alternatives

	OPTION		
	A	B	C
Buffer Space (Units)	31.0	31.0	36.0
Average Bodies (Units)	21.8	14.8	25.3
Maximum Bodies (Units)	26.0	17.0	29.0
Cycle Time (seconds)	66.5	75.0	67.1
Feasible Improvement (%)	0.2	11.0	0.2

The main conclusions are the following

- Option A and Option C show similar cycle times (between 66.5 and 67.1 seconds). The call is to disregard Option B due to subpar cycle time.
- The cycle time is higher than the required threshold of 60 seconds, both consistent with the assembly cell time, in both Option A and Option C due to:
  - Breakdowns of the assembly process
  - Breakdowns of the manual and the robot operations
- The key controlling factor is therefore the cycle time of the following assembly system. The bottleneck is at the end of the cell, which is again consistent with the theory of constraints [1].
- Option A shows a lower average number of bodies when comparing to Option C (spaces 31 vs 36; average approximately 21 vs 25), so the solution is cheaper in terms of work-in-process.
- Option A gives a feasible and viable solution with a lowest conveyor investment cost.
- Option A might be upgraded to option C if required in the future for capacity increase or additional works.

Option A is therefore selected for implementation, after the detailed and scholar simulation study that has been carried out.

6 Conclusions

The simulation study shows the weak points of the assembly process, pinpointing the necessity to address a quality assurance, balancing and improvement process. The applicability of theory-of-constraints concepts in which there has to be a synchronization between input and output flows is also highlighted. Both Options A and C also follow TOC principles by making the bottleneck be the last stage, that of the final assembly process. Work-in-process is optimum also as a by-product.

References

1. Goldratt, E.M.: The Goal. North River Press (2004)
2. Gonzalez-R, P.L., Framinan, J.M., Ruiz-Usano, R.: A Multi-Objective Comparison of Dispatching Rules In A Drum-Buffer-Rope Production Control System. International Journal of Computer Integrated Manufacturing 23(2), 155–167 (2010)
3. Johnson, N.R., Feinberg, W.E.: The Impact of Exit Instructions and Number of Exits in Fire Emergencies: A Computer Simulation Investigation. Journal of Environmental Psychology 17(2), 123–133 (1997)
4. Kim, S., Cox, J.F., Mabin, V.J.: An Exploratory Study Of Protective Inventory In A Re-Entrant Line With Protective Capacity. International Journal of Production Research 48(14), 4153–4178 (2010)
5. Kleijnen, J.P.C., Smits, M.T.: Performance metrics in supply chain management. J. Opl. Res. Soc. 54, 507–514 (2003)
6. Law, A., Kelton, W.D.: Simulation Model and Analysis. McGraw-Hill, New York (1991)
7. Patti, A.L., Watson, K.: Downtime variability: the impact of duration-frequency on the performance of serial production systems. International Journal of Production Research 48(19), 5831–5841 (2010)
8. Rhee, S.H., Cho, N.W., Bae, H.: Increasing The Efficiency Of Business Processes Using A Theory of Constraints. Information Systems Frontiers 12(4), 443–455 (2010)
9. Swain, J.: Simulation software survey (2011), `http://www.lionhrtpub.com/orms/surveys/Simulation/Simulation.html`
10. Wang, Y.H., Cao, J., Kong, L.: Hybrid Kanban/Conwip Control System Simulation and Optimization Based on Theory of Constraints. In: Proceedings of the 2009 IEEE International Conference on Intelligent Computing And Intelligent Systems, vol. 2, pp. 666–670 (2009)
11. Wu, H.H., Chen, C.P., Tsai, C.H., Yang, C.J.: Simulation and Scheduling Implementation Study of TFT-LCD Cell Plants Using Drum-Buffer-Rope System. Expert Systems With Applications 37(12), 8127–8133 (2010)
12. Yilmaz, L., Oren, T.: Agent-directed simulation and systems engineering, pp. 404–405 (2009)

Author Index