Towards a Coordination Model for Parallel Cooperative P2P Multi-objective Optimization*

M. Mezmaz, N. Melab, and E.-G. Talbi

LIFL, CNRS UMR 8022,
INRIA Futurs - Dolphin,
Université des Sciences et Technologies de Lille,
59655 - Villeneuve d'Ascq cedex - France
{mezmaz, melab, talbi}@lifl.fr

Abstract. Existing Dispatcher-Worker Peer-to-Peer (P2P) computing environments are well-suited for multi-parameter applications. However, they are limited regarding the parallel computing where the generated tasks need to communicate. In this paper, we investigate that limitation and propose a coordination model for parallel P2P multi-objective optimization (MOO). The model has been implemented on top of the XtremWeb middleware. Then, it has been experimented on a combinatorial optimization application: a parallel branch-and-bound algorithm applied to the multi-objective (MO) Flow-Shop scheduling problem. The preliminary results obtained on a network of 120 heterogeneous PCs demonstrate the efficiency of the proposed approach.

Keywords: P2P Computing, Parallelism and Coordination, Multi-objective Optimization, Branch-and-Bound, Flow-Shop.

1 Introduction

In many domains such as telecommunications, genomics, transport, and so on, the tackled optimization problems need more and more computational power. However, very often the users do not have high-end supercomputers to deal with these problems. Therefore, they have to scale down the size of the problems to solve them. In the last decade, Peer-to-Peer (P2P) computing [1] has become a real alternative to traditional supercomputing environments for the development of parallel applications that harness massive computational resources.

Nowadays, existing Dispatcher-Worker middlewares such as SETI@HOme [2], XtremWeb [3] and JNGI/JXTA [4] facilitate the development of parallel applications on P2P systems. They include a Dispatcher (server) that maintains a list of work unites and their associated data, and a set of workers (volunteer peers) that steal these work units according to the cycle stealing model. These environments all well-suited for multi-parameter applications that can be naturally

* This work is a part of the current national joint grid project GGM of ACI-Masse de Données.

split into several independent tasks. However, they are not adapted to parallel distributed applications where communications between peers are needed.

In this paper, we investigate this issue by proposing a data-driven coordination model which provides communication through a tuple space. A tuple space is a global associative memory consisting of a bag (or multi-set) of tuples. The model is dedicated to support parallel multi-objective optimization (MOO) on top of Dispatcher-Worker systems such as XtremWeb. The proposed model extends Linda [5] with group and non-blocking coordination operations that are very useful for P2P multi-objective optimization. The model has been implemented as a software layer on top of XtremWeb. At implementation level, the coordination is based on Java RMI calls.

The model has been experimented on a combinatorial optimization application: a parallel branch-and-bound algorithm applied to the bi-objective Flow-Shop scheduling problem. The problem consists in scheduling in the same order a set of jobs on a set of machines, such that the total lateness (tardiness objective) and the total completion time (makespan) are minimized. The parallelism consists in exploring in parallel a large irregular tree. The work units distributed to the workers are sub-trees to be explored. The experimentations have been conducted on a network of 120 heterogeneous PCs during more than two full days. The preliminary results demonstrate the efficiency of the proposed model and its implementation.

The rest of the paper is organized as follows: Section 2 presents a brief overview on P2P computing and coordination. Section 3 highlights the requirements of parallel cooperative MOO and then describes the proposed coordination model. Thereafter, its implementation on top of the middleware XtremWeb is discussed. Section 4 presents the experimentation of the model through the parallel Branch-and-Bound applied to the Bi-objective Flow-Shop Scheduling problem, and analyzes the preliminary experimental results. Finally, Section 5 draws some concluding remarks and the perspectives of the presented work.

2 P2P Computing and Coordination

Nowadays, there exist several fully distributed P2P systems meaning they do not include a central server [6]. These systems are often well-suited for the storage of massive data sets and their retrieval. There are also P2P systems dedicated to large scale computing [3, 2, 4, 7], but only few of them are fully distributed [7]. Fully distributed computing P2P are just emerging and are not yet mature nor stable to be exploited.

More mature software systems such as XtremWeb[3] and SETI@Home[2] are today those based on a Dispatcher-Worker architecture. In such systems, clients can submit their jobs to the Dispatcher. A set of volatile workers (peers) request the jobs from the Dispatcher according to the cycle stealing model. Then, they execute the jobs and return the results to the Dispatcher to be collected by the clients. In these middlewares, even a central server (the Dispatcher) is required for controlling the peers (workers) they are considered as P2P software environments. Indeed, an important part of these systems is executed on these peers with a high autonomy. In addition, a hierarchical design allows them to deal with a larger number of peers.

One of the major limitations of P2P computing environments is that they are well-suited for embarrassingly parallel (e.g. multi-parameter) applications with independent tasks. In this case, no communication is required between the tasks, and thus peers. The deployment of parallel applications needing cross-peer/task communications is not straightforward. The programmer has the burden to manage and control the complex coordination between the workers. To deal with such issue existing middlewares must be extended with a software layer which implements a coordination model. Several interesting coordination models have been proposed in the literature [8]. In this paper, we focus only on one of the most popular of them i.e. Linda [5] because our proposed model is inspired from that model.

In the Linda model, the coordination is performed through generative communications. Processes share a virtual memory space called a *tuple-space* (set of tuples). The fundamental data unit, a tuple, is an ordered vector of typed values. Processes communicate by reading, writing, and consuming these tuples. A small set of four simple operations allows highly complex communication and synchronization schemes:

– *out(tuple)*: puts *tuple* into *tuple-space.*
– *in(pattern)*: removes a (often the first) tuple matching *pattern* from *tuple-space.*
– *rd(pattern)*: is the same as *in(pattern)*, but does not remove the tuple from *tuple-space.*
– *eval(expression)*: puts *expression* in *tuple-space* for evaluation. The evaluation result is a tuple left in *tuple-space.*

Due to the high communication delays in a P2P system, tuple rewriting is very important as it allows to reduce the number of communications and the synchronization cost. Indeed, in Linda a rewriting operation is performed as an “in” or “rd” operation followed by a local modification and an “out” operation. The operations “in”/“rd” and “out” involve two communications and an heavy synchronization. Therefore, a rewriting (or update) operation is very useful for coordination in P2P environments.

3 The Proposed P2P Coordination Model

3.1 Parallel MOO and Coordination

A multi-objective problem (MOP) consists generally in optimizing a vector of nb_{obj} objective functions $F(x) = (f_1(x), \ldots, f_{nb_{obj}}(x))$, where x is an d-dimensional decision vector $x = (x_1, \ldots, x_d)$ from some universe called *decision space.* The space the objective vector belongs to is called the *objective space.*

F can be defined as a cost function from the decision space to the objective space that evaluates the quality of each solution $(x_1, \ldots, x_d)$ by assigning it an objective vector $(y_1, \ldots, y_{nb_{obj}})$, called the *fitness*.

Unlike single-objective optimization problems, a MOP may have a set of solutions known as the *Pareto optimal set* rather than an unique optimal solution. The image of this set in the objective space is denoted as *Pareto Front or PF*). Graphically, a solution x is Pareto optimal if there is no other solution x' such that the point $F(x')$ is in the dominance cone of $F(x)$. This dominance cone is the box defined by $F(x)$, its projections on the axes and the origin (Fig. 1).

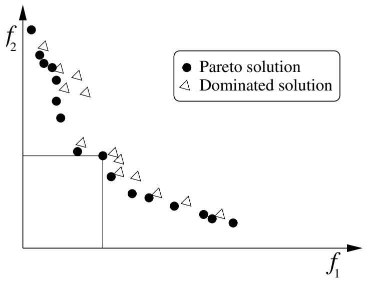

Fig. 1. Example of Pareto solutions

There are two major categories of MOO methods: MO exact methods and MO meta-heuristics. While the first category allows to find optimal solutions for MOPs, the second class provides near-optimal solutions in a reasonable time. Real-world MOPs involve highly constrained design and and high computational cost. Parallelism is proved to be a powerful way to achieve efficiency and effectiveness. In general, parallel MO exact methods consist in exploring in parallel a search three. The processes share the best found PF, which is remotely updated each time a process has discovered in its search sub-tree a better PF. Coordination operations are required to update this shared information.

In parallel models of MO meta-heuristics [9] such as the island model, different processes cooperate by exchanging Pareto optimal solutions in order to improve the effectiveness. The exchange may be performed either directly by message passing or through a shared space. In the last case, a coordination model is required to ensure the cooperation.

3.2 The Coordination Model

Designing a coordination model for parallel MOO requires the specification of the content of the tuple space, a set of coordination operations and a pattern matching mechanism. According to the comments of the previous sub-section, the tuple space may be composed of a set of Pareto optimal solutions and their corresponding solutions in the objective space. For the parallel exact MO methods, all the solutions in the tuple space belong to the same PF i.e. the actual best found one. For the parallel island model of the MO meta-heuristics, the tuple space contains a collection of (parts of) Pareto optimal sets deposited by the islands for migration. The mathematical formulation of the tuple space (*Pareto Space or PS*) is the following:

$$PS = \bigcup PO,\ with\ PO = \{(x, F(x)),\ x\ is\ Pareto\ optimal\}$$

In addition to the operations provided in Linda, parallel P2P MOO needs other operations. These operations can be divided in two categories: *group* operations and *non-blocking* operations. Group operations are useful to manage multiple Pareto optimal solutions. *Non-blocking* operations are necessary to take into account the volatile nature of P2P systems. In our model, the coordination primitives are defined as follows:

– *in, rd, out and eval*: These operations are the same as those of Linda defined in Section 2.
– *ing(pattern)*: Withdraws from PS all the solutions matching the specified pattern.
– *rdg(pattern)*: Reads from PS a copy of *all* the solutions matching the specified pattern.
– *outg(setOfSolutions)*: Inserts multiple solutions in PS.
– *update(pattern, expression)*: Updates *all* the solutions matching the specified pattern by the solutions resulting from the evaluation of *expression*.
– *inIfExist, rdIfExist, ingIfExist and rdgIfExist*: These operations have the same syntax than respectively *in, rd, ing and rdg* but they are *non-blocking* probe operations.

The *update* operation allows to locally update the Pareto space, and so to reduce the communication and synchronization cost. The pattern matching mechanism depends strongly on how the model is implemented, and in particular on how the tuple space is stored and accessed. For instance, if the tuple space is stored in a database the mechanism can be the request mechanism used by the database management system. More details on the pattern matching mechanism of our model are given in the next Section.

3.3 Implementation on Top of XtremWeb

XtremWeb [3] is a Java P2P project developed at Paris-Sud University. It is intended to distribute applications over a set of peers, and is dedicated to multi-parameter applications that have to be computed several times with different inputs. XtremWeb manages tasks following the Dispatcher-Worker paradigm. Tasks are scheduled by the Dispatcher to workers only on their specific demand since they may adaptively appear (connect to the Dispatcher) and disappear (disconnect from the Dispatcher). The tasks are submitted by either a client or a worker, and in the latter case, the tasks are dynamically generated for parallel execution. The final or intermediate results returned by the workers are stored in a MySQL database. These results can be requested later by either the clients or the workers. The database stores also different information related to the workers and the deployed application tasks.

XtremWeb is well-suited for embarrassingly parallel applications where no cross-peer communication occurs between workers, and these can only communicate with the Dispatcher. Yet, many parallel distributed applications particularly parallel MOO ones need cooperation between workers. In order to free the user from the burden of managing himself or herself such cooperation we propose an extension of the middleware with a software layer.

The software layer is an implementation of the proposed model composed of two parts: a coordination API and its implementation at the worker level and a coordination request broker (CRB). The Pareto Space is a part of the MySQL database associated with the Dispatcher. Each tuple or solution of the Pareto Space is stored as a record in the database. From the worker side the coordination API is implemented in Java and in C/C++. The C/C++ version allows the deployment and execution of C/C++ applications with XtremWeb (written in Java). The coordination library must be included in these programmer applications. From the Dispatcher side, the coordination API is implemented in Java as a Pareto Space manager. The *CRB* is a software broker allowing the workers to transport their coordination operations to the Dispatcher through RMI calls.

4 Application to Parallel MO Branch-and-Bound

In this Section, we describe the sequential B&B algorithm, then a parallel version using the proposed coordination model, and finally some experiments performed on a P2P network through the Flow Shop problem.

4.1 Parallel MO Branch-and-Bound

MO Branch-and-bound algorithms (MO-B&B) solve MOPs by iteratively partitioning the solution space into subspaces (each subspace is associated with a sub-MOP). In this paper, we assume that the MOP to solve is a minimization MOP. A sequential MO-B&B algorithm consists in iteratively applying five basic operations over a list of problems: *Branching*, *Resolution*, *Bounding*, *Selection* and *Elimination*.

Successive branching (decomposition) operations create a tree of MOPs rooted in the original MOP. The value of the best PF found so far is used to prune the tree and eliminate the MOPs that are likely to lead to worse solutions. At each step, a MOP is selected either according to the bound values (as in *best-first* strategy) or not (as in *depth-first* and *breadth-first* strategies). The selected MOP may not be split because it has no solution or because a solution is already be found. In this case, it is solved, and if its solution can improve the best known PF this latter is updated. If the MOP can be split than it is decomposed into smaller sub-MOPs. A sub-MOP is eliminated if its bound value is not better that the best known PF. Otherwise, it is added to the pool of MOPs to be solved.

There exist different parallel models in B&B algorithms [10]. We focus here on the most general approach, in which the B&B tree is built in parallel by performing simultaneously the operations presented above on different sub-MOPs. According to such approach, the design of parallel B&B algorithms is based on three major parameters: the *execution mode*, the *work sharing strategy* and the *information sharing policy.* The execution mode defines what processes do after completion of a work unit, and may be *synchronous* or *asynchronous.* The processes (do not) wait for each other in a(n) (a)synchronous mode. The work sharing strategy defines how work units are assigned to processes to efficiently exploit available parallelism. The information sharing policy indicates how the best-known solution is published and updated.

In this paper, we propose an asynchronous parallel cooperative Dispatcher-Worker MO B&B algorithm. Asynchrony is required by the heterogeneity nature of the P2P target execution architecture. The work sharing strategy follows the idle cycle or work stealing paradigm. Each worker maintains its local pool of MOPs to be solved. When the local pool is empty, the worker sends a work request to the Dispatcher. The information sharing issue is solved as the following: the best known PF is published and maintained by the Dispatcher. This information is requested by the workers, and updated each time a better PF is locally found.

At each step, the worker tests if there is some MOP to solve in its local work pool. If the pool is empty it requests work from the Dispatcher. The Dispatcher replies with a pool of work units and the value of the best-known PF. This value is stored locally. Otherwise, if there is some work in the local pool, the worker performs a step of the sequential MO-B&B on its local pool. Thereafter, it probably requests the Dispatcher to update the best-known PF by merging this latter with its local version. The operation is performed by the Pareto Space Manager on the Dispatcher. The new best-known PF is returned to the calling worker.

4.2 Application to the Flow-Shop MOP

The Flow-Shop MOP is one of the numerous scheduling MOPs [11] that has received a great attention given its importance in many industrial areas. The MOP can be formulated as a set of N jobs $J_1, J_2, \ldots, J_N$ to be scheduled on M machines. The machines are critical resources as each machine can not be simultaneously assigned to two jobs. Each job J_i is composed of M consecutive tasks $t_{i1}, \ldots, t_{iM}$, where t_{ij} represents the j^{th} task of the job J_i requiring the machine m_j. To each task t_{ij} is associated a processing time p_{ij}, and each job J_i must be achieved before a due date d_i.

The MOP being tackled here is the Bi-objective Permutation Flow-Shop MOP (*BPFSP*) where jobs must be scheduled in the same order on all the machines. Therefore, two objectives have to be minimized: (1) C_{max}: Makespan (Total completion time), (2) T: Total tardiness. The task t_{ij} being scheduled at time s_{ij}, the two objectives can be formulated as follows:

$$f_1 = C_{max} = Max\{s_{iM} + p_{iM} | i \in [1 \ldots N]\}$$
$$f_2 = T = \sum_{i=1}^{N} [max(0, s_{iM} + p_{iM} - d_i)]$$

In this paper, we do not focus on how the MO-B&B technique is applied to BPFSP, the reader is referred to [12] for such details. We are interested in the parallel P2P design features. In the implementation of the parallel MO-B&B applied to BPFSP, the best-known PF is updated if a sufficient number of iterations is already performed. The adopted selection strategy is the *depth-first* one, the node with the best bound being chosen at each step. The *update* coordination operation is executed on the Dispatcher by the Pareto Space Manager. First, it consists in performing an union between the two sets: the global best-known PF (stored in the Pareto Space) and its local version. The new best-known PF is then selected from this union set by considering all the non-dominated solutions. The new result is returned to the calling Worker.

After a fixed number of iterations, if a Worker has work in its local pool it splits it into as many pools as available workers (considering itself). The Worker saves a pool for its own need, and submits (in a client role) the other pools to the Dispatcher through the *eval* coordination operation. The Dispatcher puts these work units in its task pool to be sent to available workers at their request.

4.3 Experimentation

The application has been deployed on the education network of the *Polytech'Lille* engineering school. The experimentation hardware platform is composed of 120 heterogeneous Linux Debian PCs. The BPFSP MOP benchmark is composed of 10 jobs and 20 machines ($n = 10$ and $m = 20$). The experimental results are summarized in Table 1. The total execution time is measured for the sequential and parallel versions. The execution time of the sequential version is normalized as the whole target architecture is heterogeneous. The machine where the sequential algorithm is executed is an Intel Pentium 4, 3 GHz, and is considered as a reference machine for the computation of the normalized factor. The normalized factor for each peer is obtained by using an application specific benchmark task that is sent to all workers that join the computational pool. The speed at which the benchmark task is completed is considered as the normalized factor.

Formally, let t_{ref} and t_i be the execution time of the benchmark on respectively the reference machine and the machine number $i = 1..N$ of the pool of worker peers. The normalized factor α_i associated with the worker peer i is computed as follows: $\alpha_i = \frac{t_i}{t_{ref}}$. Let α_{av} be the average normalized factor for all the worker peers. It can be formulated as: $\alpha_{av} = \frac{\sum_{i=1}^{N} \alpha_i}{N}$. The sequential time reported in Table 1 is the time obtained on an average peer, obtained by multiplying the sequential time obtained on the reference peer by the average factor.

The results show that the execution time is divided by over 29 on 120 machines. One has to note that the experiments have been performed during a working day, thus the experimentation environment is non-dedicated. On the

Table 1. Parallel MO-B&B vs. Sequential MO-B&B

	Sequential B&B	**Parallel B&B**
Total number of tasks	1	657
Total execution time	54h51	1h53

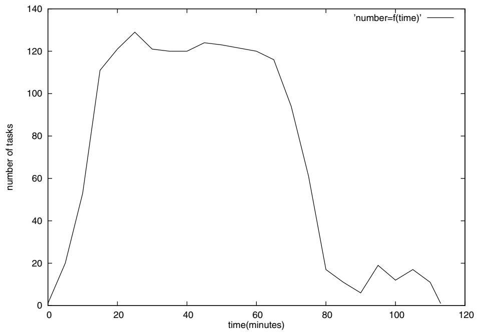

Fig. 2. Task generation over time

other hand, as Fig. 2 illustrates it, due to the nature of the application sufficient parallelism (for the 120 peers) is generated during only 1 hour over about two hours of total execution.

5 Conclusion and Future Work

In this paper, we have presented a coordination model for parallel cooperative MOO applications in a P2P environment. The model allows to overcome a major limitation of existing Dispatcher-Worker middlewares: they do not allow a straightforward communication between tasks executed by different peers. The model has been implemented on top of XtremWeb, a middleware dedicated to the execution of independent multi-parameter applications. The result is that the users can develop parallel cooperative applications in a transparent way. They do not need to manually manage and control the cooperation. Furthermore, the model is generic and can be integrated into another P2P computing middleware and applied in the context of another application domain.

The model has been experimented and validated on an MOO application: a parallel B&B algorithm applied to the Bi-criterion Permutation Flow-Shop Scheduling problem. The experimental results show that the time wasted by the PCs of the experimentation hardware platform is well exploited as the total execution time of the application is divided by a factor of 4 in a non-dedicated execution environment.

In the future, we will experiment and extend the model to deal with parallel cooperative meta-heuristics. We will also deploy it on top of another middleware (JNGI/JXTA [4]) to demonstrate its generic nature.

References

1. Oram, A.: Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates (2001)
2. Anderson, D., Cobb, J., Korpela, E., , Lepofsky, M., Werthimer, D.: SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM **Vol. 45** (2002) 56–61
3. Fedak, G., Germain, C., Neri, V., Cappello, F.: XtremWeb: building an experimental platform for Global Computing. Workshop on Global Computing on Personal Devices (CCGRID2001), IEEE Press (2001)
4. Verbeke, J., Nadgir, N., Ruetsch, G., Sharapov, I.: Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. In Proc. of the Third Intl. Workshop on Grid Computing (GRID'2002), Baltimore, MD (2002) 1–12
5. Gelernter, D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems **Vol. 7** (1985) 80–112
6. Ripeanu, M.: Peer-to-Peer Architecture Case Study: Gnutella Network. 1^{st} IEEE Intl. Conf. on Peer-to-peer Computing (P2P2001) (2001)
7. Oliveira, L., Lopes, L., Silva, F.: P^3: Parallel Peer to Peer - An Internet Parallel Programming Environment. Intl. Workshop on Peer-to-Peer Computing, Pisa, Italy (2002)
8. Papadopoulos, G., Arbab, F.: Coordination models and languages. Advances in Computers: The Engineering of Large Systems, Academic Press **46** (1998)
9. van Veldhuizen, D., Zydallis, J., Lamont, G.: Considerations in engineering parallel multiobjective evolutionary algorithms. IEEE Trans. on Evolutionary Computation **7** (2003) 144–173
10. Gendron, B., Crainic, T.: Parallel branch-and-bound algorithms: Survey and synthesis. Operations Research **42** (1994) 1042–1066
11. T'kindt, V., Billaut, J.C.: Multicriteria Scheduling - Theory, Models and Algorithms. Springer-Verlag (2002)
12. Basseur, M., Lemesre, J., Dhaenens, C., Talbi, E.G.: Cooperation between Branch and Bound and Evolutionary Approaches to solve a BiObjective Flow Shop Problem. Worshop on Evolutionary Algorithms (WEA'04) (2004) 72–86