Task Delegation in a Peer-to-Peer Volunteer Computing Platform

Kristóf Attila Horváth[1] and Miklós Telek[1,2(✉)]

[1] Budapest University of Technology and Economics, Budapest, Hungary
horvath.kristof.attila@gmail.com
[2] MTA-BME Information systems research group, Budapest, Hungary
telek@webspn.hit.bme.hu

Abstract. The paper reports an effort made for understanding the effect of task delegation policy in a peer-to-peer volunteer computing platform. This effort includes the implementation of a simulation environment and the development of associated analytical models for the analysis of task delegation policies in peer-to-peer computing platforms. Based on the analytical model best and worst task delegation policies are computed and the resulted system behavior is verified by simulation.

Keywords: Peer-to-peer volunteer computing platform · Task delegation · Mean field model · Simulation

1 Introduction

The concept of utilizing the unutilized computing resources of a large number of (personal) computers connected via the internet is around for several decades. There are widely known peer-to-peer volunteer computing platform projects established for evaluating various computationally intensive tasks (a summary is provided in the next section). The related literature discusses the introduction, the spread, the order of magnitude, the organization and the applied technical details of these projects. In this work we focus on a particular detail of the organization of peer-to-peer volunteer computing platforms, the subtask delegation policy.

As the organization of volunteer computing platform changes from centrally controlled to peer-to-peer based, by time it became important to understand the performance consequences of autonomous subtask delegation policies.

The rest of the paper is organized as follows. Section 2 introduces the existing computing platforms and the related literature. We summarize the main properties of a proposed peer-to-peer volunteer computing platform in Section 3. Analytical models and associated performance analysis of various parameters of interests are investigated in Section 4. Finally, Section 5 presents the simulation results and their relation to the results of the analytical models.

The authors thank the support of the OTKA K101150 project.

M. Gribaudo et al. (Eds.): ASMTA 2015, LNCS 9081, pp. 115–129, 2015.
DOI: 10.1007/978-3-319-18579-8_9

2 Existing Volunteer Computing Platform Solutions

2.1 A Brief History of Volunteer Computing

There is a huge amount of unused computing capacity in personal computers, because the computers of users work 100% occupancy only negligible part of the time. This was the basis of the volunteer computing networks which utilize the unused capacity of personal computers. A study was published about the capacity of volunteer computing networks in 2006 [4], despite of old data the measured values are shocking: an ordinary volunteer computing project could use 95.5 teraFLOPS (10^{12} Floating-point Operations Per Second) computing capacity and 7.74 petabyte (10^{15} byte) storage.

The first volunteer computing projects started in 1997: the GIMPS (Great Internet Mersenne Prime Search) and the Distributed.net where cryptographic algorithms were tested. These projects had got tens of thousands of volunteer users [2].

The first project, which already had got millions of volunteer users, is the SETI@home project. It has started in 1999. SETI is the abbreviation of Search for Extraterrestrial Intelligence, and the @home (at home) suffix refers the use of personal computers instead of supercomputers. Tiny pieces of received signals of radio telescopes were sent to the computers of volunteers where the client application tried to find very narrowband (<Hz) signals in them. The method assumes that the extraterrestrial intelligence transmits narrowband signal which is easily distinguishable from the natural background radiation. The task is highly computationally intensive because a lot of parameters – bandwidth, symbol duration time, Doppler shift, etc. – are unknown [6,13].

Because of the popularity of SETI@home project a general platform called BOINC (Berkeley Open Infrastructure for Network Computing) was developed in 2002. The BOINC platform became dominant in the subsequent years.

In the volunteer computing projects one of the hardest challenge is finding and keeping members as volunteers. Spectacular figures of the scientific results in wallpaper or in screen saver try to increase the interest. An other option is to publish the list of most effective volunteers and [17] recommends worker teams to utilize the team spirit. In spite of the seemingly infinite resources the performance optimization of distributed computing platforms is an essential goal [1].

2.2 Platforms

BOINC was developed at the University of California, Berkeley. It is the largest volunteer computing platform so far. The projects of the platform are computed on 600 000 personal computers. The total computing capacity almost reaches 10 petaFLOPS, therefore the system rivals the most powerful supercomputers [19]. Apart of the SETI@home project the platform hosts additional projects like Einstein@home, LHC@home, Milkyway@home, etc [3,5].

The XtremWeb platform was developed in parallel and independently from the BOINC system. The objective and the implementation are very similar in the two platforms, however in the competition for users BOINC was more successful [11].

Alchemi is a .NET-based platform, which was developed at the University of Melbourne. In this system the main objective is the easy programmability, the other aspects are less important [15,16].

The OurGrid platform is based on a new idea. This platform interconnects the grid systems of universities and research groups intend to utilize the free resources [7,8].

All of these platforms follow the master–worker parallel programming paradigm. The central server decomposes the task into subtasks and manages the delegation. The lifecycle of a subtask is the following: (a) the server creates a job by packing the executable code and the input files together (b) the client downloads the job (c) the client computes the results (d) the client uploads the results (e) the server verifies and processes the results.

3 Properties of the Proposed Distributed Computing Platform Solution

The existing volunteer computing platforms have got two main issues. The first one is the protection of the volunteer's computer. In BOINC and XtremWeb the servers send native executable code to the clients. The platforms use asymmetric cryptography to ensure the authenticity and the integrity, but the project owner can execute anything on the volunteer's computer.

On the other hand Alchemi and OurGrid systems use virtualization to solve the problem, but it reduces the performance which should be avoided in a computing platform. In the proposed computing platform the elements send the source code to the peers. This method includes filtering of malicious codes. The compiling–running combination may be more efficient than the virtualization.

The second issue is that the management of subtasks is centralized in all existing platforms; they follow the master–worker programming paradigm. In some systems the executable code or input files can be shared by peer-to-peer mechanisms [9,10], however the central management of subtasks are presented here as well.

In the proposed distributed volunteer computing platform, every node can delegate subtasks to other nodes, if the currently computed subtask contains parallel blocks. In this approach the programmer can write multi-level subtask structures, so a subtask in a parallel block can be the same as the main program following a fractal-like structure. The nodes in the computing network are identical, so a homogenous programming model can be used instead of a heterogenous programming model as in CUDA [12], OpenCL [18], etc.

The codes can contain serial- and parallel blocks sequentially. The parallel blocks must be fully decomposable, so the subtasks can not communicate with each other or with the main task (except the interchange through input- and output files). The next serial- or parallel block can be started after all subtasks of the current block had been completed.

At the beginning of the parallel blocks the client program decides how many subtasks will be computed locally and how many will be delegated. Because of

the overhead of subtask delegation there is a trade-off between locally computed and delegated subtasks. The optimal strategy is investigated and the behavior of the whole system is analysed in the following section.

4 Performance Analysis of Distributed Computing Platforms

In this section we investigate the performance of distributed computing platforms with different analysis approaches. The two main applied analytical approaches are the phase type (PH) distributions and the mean field approximations, which we also summarize below.

4.1 Phase Type Distributions

If the stochastic behavior of a real system can be characterized by a Markov chain then various random event times which are of practical interests are PH distributed. This statement applies for both discrete and continuous time Markov chains (DTMCs and CTMCs) with associated discrete or continuous PH distributions. In this work we focus on continuous time models.

Definition 1. *The time to reach the absorbing state in a CTMC with n transient and an absorbing state is (size n)* phase type *distributed.*

Consequently, a (continuous) PH distributed random variable $\mathcal{X}$ is continuous non-negative with cumulative distribution function

$$F(t) = Pr(\mathcal{X} < t) = 1 - \underline{v} e^{\boldsymbol{H}t} \mathbb{1} \ ,$$

where row vector $\underline{v}$ contains the initial probabilities of the CTMC in the transient states, square matrix $\mathbf{H}$ contains the transition rates among the transient states and column vector $\mathbb{1}$ is composed by ones. $\underline{v}$, $\mathbf{H}$ and $\mathbb{1}$ are referred to as initial probability vector, transient generator matrix and closing vector, respectively. Throughout the paper we assume that the Markov chain starts from a transient state, i.e., $\underline{v}\mathbb{1} = 1$, and consequently $\mathcal{X}$ has no probability mass at zero. The density and the moments of $\mathcal{X}$ are

$$f(t) = \underline{v} e^{\boldsymbol{H}t} (-\mathbf{H}) \mathbb{1} \ , \tag{1}$$

$$\mu_n = E(\mathcal{X}^n) = n! \underline{v} (-\mathbf{H})^{-n} \mathbb{1} \ . \tag{2}$$

If the initial probability vector and the transient generator matrix are obtained from modeling assumptions all performance measures associated with $\mathcal{X}$ can be computed based on (1) and (2).

4.2 Execution Time Model

The execution time of a parallel block can be analyzed by the CTMC shown in Figure 1. The states can be arranged in a two-dimensional grid: horizontal axis shows the number of locally computed subtasks, the vertical axis shows the number of delegated subtasks, which are computed on other computers.

The absorbing state, which represents the completion of a task, is state $(0, 0)$. The CTMC in Figure 1 describes the case when a subtask can be computed locally in an exponentially distributed amount of time with parameter μ_1, and a delegated subtask can be sent, computed and returned in an exponentially distributed amount of time with parameter μ_2.

From any states (except the states at the top and the left boundaries) there are two possible transitions:

1. a locally-computed subtask completes with rate μ_1 and the process goes from state (i, j) to state $(i, j-1)$,
2. one of the i delegated subtasks arrives with rate $i\mu_2$ and the process goes from state (i, j) to state $(i-1, j)$.

There is no local (dedicated) computation at the left (top) boundary, so there is only one of the two transitions at these boundaries.

Based on the transition graph the infinitesimal generator matrix of the process $\mathbf{Q}$ is obtained by mapping the nodes of the transition graph, which are the states of the Markov chain, to a subset of natural numbers and indicating the transition rates between the pair of states. The transient generator, matrix $\mathbf{H}$ in (1) and (2), contains the transition rates only among the transient states, and is the lower right block of matrix $\mathbf{Q}$ as it is indicated below for the state space with 3×3 states.

$$\mathbf{Q} = \left[\begin{array}{c|cccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline \mu_1 & -\mu_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_1 & -\mu_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \mu_2 & 0 & 0 & -\mu_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_2 & 0 & \mu_1 & -\mu_1-\mu_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_2 & 0 & \mu_1 & -\mu_1-\mu_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu_2 & 0 & 0 & -2\mu_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu_2 & 0 & \mu_1 & -\mu_1-2\mu_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu_2 & 0 & \mu_1 & -\mu_1-2\mu_2 \end{array}\right] \tag{3}$$

The elements under the diagonal contain μ_1 except every i^{th}, which is 0, because local computing does not happen at the first column of the grid. The i^{th} elements under the diagonal contain the rates of arriving delegated task, which are μ_2, $2\mu_2$, etc. in i-length blocks. Conventionally, the diagonal elements contain the negation of sum of all the other elements in the row.

Expected Value of the Task Execution Time. The mean task execution time (the running time of parallel block), which is the mean time to reach state

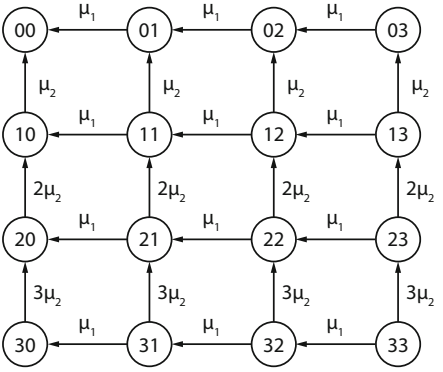

Fig. 1. Phase type CTMC for execution time modeling

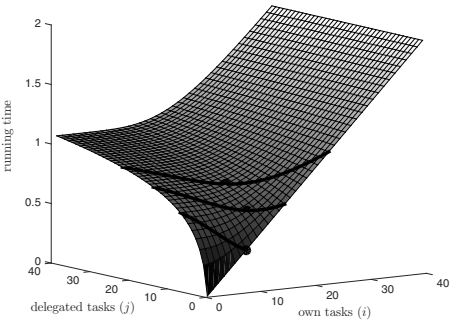

Fig. 2. Running times for different states

$(0,0)$ in the Markov chain in Figure 1, is an essential performance measure of the distributed computing platform in order to optimize the delegation strategy. In general, the mean task execution time can be computed based on (2), but utilizing the structural properties of matrix $\mathbf{H}$ it can also be computed by an efficient recursive procedure.

If the process is in state (i,j), $i>0$, $j>0$, it can move forward to one of the two possible consecutive states. On both cases we can compute the probability of taking one of the two possible consecutive states and the expected time to reach the $(0,0)$ state from the next state. If the local calculation completes sooner and the process goes on the first trajectory we have:

$$E(\hat{T}_{i,j}) = \frac{1}{\mu_1 + i\mu_2} + E(T_{i,j-1}) \ . \tag{4}$$

If one of the delegated tasks completes sooner and the process goes on the second trajectory:

$$E(\check{T}_{i,j}) = \frac{1}{\mu_1 + i\mu_2} + E(T_{i-1,j}) \ . \tag{5}$$

To calculate the full time, the two conditional expected times, (4) and (5), have to be weighted by the probabilities of the trajectories.

$$\begin{aligned} E(T_{i,j}) &= \Pr(1^{\text{st}} \text{ trajectory}) E(\hat{T}_{i,j}) + \Pr(2^{\text{nd}} \text{ trajectory}) E(\check{T}_{i,j}) \\ &= \frac{\mu_1}{\mu_1 + i\mu_2} \left(\frac{1}{\mu_1 + i\mu_2} + E(T_{i,j-1}) \right) + \frac{i\mu_2}{\mu_1 + i\mu_2} \left(\frac{1}{\mu_1 + i\mu_2} + E(T_{i-1,j}) \right) \ . \end{aligned} \tag{6}$$

This recursive formula is valid for every state where $i>0$ and $j>0$. The mean task execution time from the boundary states where $i=0$ or $j=0$ can be computed as follows. If the phase type distribution is initialized from a state where $i=0, j>0$, the task execution time is Erlang(j,μ_1) distributed (the sum

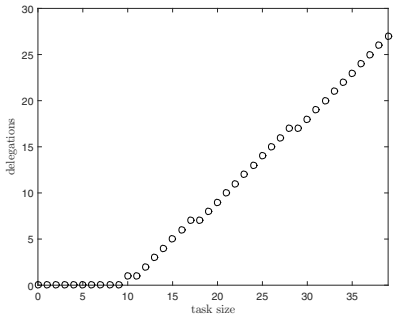

Fig. 3. Optimal number of delegated subtasks

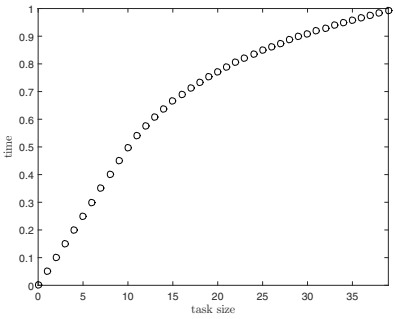

Fig. 4. Optimized running time

of j independent exponentially distributed random times with the parameter μ_1), whose expected time is

$$E(T_{0,j}) = j\frac{1}{\mu_1}\ . \tag{7}$$

In case of $j = 0, i > 0$ the successive exponential phases do not have the same parameter, so the expected values of the phases with phase dependent rates ($k\mu_2$, $k = i, i-1, \ldots, 1$) have to be summed up.

$$E(T_{i,0}) = \sum_{k=1}^{i} \frac{1}{k\mu_2} = \frac{1}{\mu_2}\sum_{k=1}^{i}\frac{1}{k} = \frac{1}{\mu_2}H_i, \tag{8}$$

where $H_i = \sum_{k=1}^{i}\frac{1}{k}$ is commonly referred to as the i^{th} harmonic number.

Optimalization. To investigate the optimal subtask delegation we evaluate the mean task execution time from different initial states. Figure 2 depicts the mean task execution time as a function of the initial state, when $\mu_1 = 20$ and $\mu_2 = 4$. Indeed the continuous surface on Figure 2 is valid only at integer points.

If the overhead of delegation reduces and the ratio of two intensities converges to zero, then the surface would converge to the $E(T) = i\mu_1$ plane.

When the program reaches a parallel block, it has to decide how many subtasks will be computed locally and how many will be delegated. This two numbers determine the starting state of the process. When a parallel block is composed by N subtasks the starting state satisfies the $i + j = N$ formula, therefore the optimum is the minimum of the surface on the $i + j = N$ section plane. The section plane has got N integer points, the optimal running time is obtained at the minimum of these. Figure 2 contains the $N = i + j = 8, 15, 23$ section plane on the surface and their minimum points. These points indicate the optimal number of delegated subtask in case of n parallel tasks.

Figure 3 shows the optimal number of delegated subtasks as a function of the total number of subtasks. Up to a threshold (10 in the example) every subtask is computed locally, after that almost all of the following subtasks are delegated.

Figure 4 shows the optimal running time as a function of the total number of subtasks. Up to the threshold where the optimal execution time is obtained with no subtask delegation ($N < 10$) the plot is linear with slope $1/\mu_1$, which is in line with formula (7). After that point the slope breaks down due to the effect of delegation as it is visible from a comparison with Figure 3. According to (8) the mean time to complete a number of delegated task is related to the harmonic series. The i^{th} harmonic number is approximately equal to $\ln(i)$, because $\int_1^n \frac{1}{x} dx = \ln(n)$ and the harmonic series is an approximation to the definite integral. The difference between the harmonic series and the logarithmic series converges to the Euler–Mascheroni constant: $\lim_{n\to\infty} H_n - \ln(n) = \gamma \approx 0.5772156649$.

Model Parameters Setting. The expected value of running time of a locally-computed subtask is the reciprocal of μ_1, so μ_1 can be defined as the FLOPS of the machine divided by the floating-point operations of a subtask. Similarly, the reciprocal of μ_2 can be defined as the sum of the floating-point operations of one subtask divided by the average FLOPS in the computing network and the mean time to transmit the input/output data through the internet connection.

The above described Markovian model and this parameters setting do not guarantee the perfect matching of the model and the real execution times in the computing platform. For this reason we also verify the results by simulation in Section 5.

4.3 Mean Field Approximation

In the previous subsections we optimized the system behavior assuming infinite computing resources. To consider the effect of finite computing resources we apply a different modeling approach, the mean field approximation. The mean field method allows the analysis of large Markov systems which are composed by a finite number of identical interacting components, where the interaction depends only on the number of components which are in particular states. For example, in our case the identical components are the computing units and the dependence of one component on the other components is only through the number of available idle computing units. Let N be the number of components. The state of component ℓ ($\ell = 1, 2, \ldots, N$) at time t is denoted by $X_\ell(t)$. In our case the state of the component depends on the task it is working on. A component could be idle, working on a delegated task, and being responsible for the execution of the task composed by parallel subtasks. The latest case can be described by a state space similar to the one on Figure 1. The overall behavior of these three possible cases is discussed below and depicted on Figure 5. The state space of each component, S, is composed by $s = |S|$ states, and $N_i(t)$ denotes the number of components which are in state i ($\forall i \in S$) at time t. For example,

$N_{idle}(t)$, denotes the number of idle computing units which are available for task or subtask assignment. In our case it is intuitive to see that a delegation decision depends only on the number of idle computing units and not on the state of a particular computing unit of the system. The row vector composed by $N_i(t)$ is denoted by $\mathbf{N}(t)$ and by this definition, $\sum_{i=1}^{s} N_i(t) = N$ (is the number of all computing units in our case).

The global behavior of the set of N components forms a CTMC over the state space of size s^N. However, due to the fact that the components are identical and indistinguishable, the state space can be lumped into the aggregate state space S_L of size $\binom{N+s-1}{s-1}$, where a state of the overall CTMC is identified by the number of components staying in each state of S, i.e., by $\mathbf{N}(t) = (N_1(t), N_2(t), \ldots, N_s(t))$ (in our case it means the number of idle components, the number of components working on a delegated task, the number of components responsible for the computation of a task and waits for the completion of two delegated subtasks, etc). $\mathbf{N}(t)$ refers to the population vector, which describes the distribution of the population between the possible states.

The evolution of a given computing unit is such that the transition rates may depend on the global behavior through the actual value of vector $\mathbf{N}(t)$. With this assumption, the transition rate of a particular component from state i to j is $K_{ij}(\mathbf{N}(t))$ which depends only on vector $\mathbf{N}(t)$. The diagonal elements of the transition rate matrix are defined by $K_{ii}(\mathbf{N}(t)) = -\sum_{j \in S, j \neq i} K_{ij}(\mathbf{N}(t))$.

Instead of using the population vector, $\mathbf{N}(t)$, the normalized population vector, $\mathbf{n}(t) = \mathbf{N}(t)/N$ is commonly used for the mean field analysis of such systems. The entries of $\mathbf{n}(t)$ define the proportion of objects in state i at time t and $\sum_{i \in S} n_i(t) = 1$. The associated transition rate function is denoted by $k_{ij}(\mathbf{n}(t))$, and the matrix composed by these elements is $\mathbf{k}(\mathbf{n}(t)) = \{k_{ij}(\mathbf{n}(t))\}$. Hereafter we assume that $k_{ij}(\mathbf{n}(t))$ is a Lipschitz continuous function over the s-dimensional unit cube. The mean field method is based on the following essential theorem.

Theorem 1. *[14] The normalized state vector of the population process, $\mathbf{n}(t)$, tends to be deterministic, in distribution, as N tends to infinity and satisfies the following differential equation*

$$\frac{d}{dt}\mathbf{n}(t) = \mathbf{n}(t)\ \mathbf{k}(\mathbf{n}(t))\ . \tag{9}$$

The mean field approximation is based on the fact that for large but finite N the deterministic approximation according to (9) is a good approximation of the system behavior with well defined error bounds [14].

4.4 The Mean Field System Model

The model in Section 4.2 optimizes the task execution time disregarding its effect on resource utilization. In this section, we present an approximate mean field model of the system which considers also the resource utilization.

When the studied volunteer computing project runs a task with one level delegation policy (delegated subtasks are not divided into lower level subsubtasks), then the state space of a computing device is similar to the state space of the execution time model when the computing device is responsible for the computation of a task composed of a given number of subtasks. In this case state (i, j) is the state when the computing device is waiting for the completion of i delegated subtasks and the node has to compute j subtasks locally. In state $(0, 0)$ the node is idle, so it can start with a new task or it can receive a delegated subtask from another node. Additional to the states associated with the task computation there is state $(*)$ which identifies the state when the node computes a delegated subtask.

To describe the considered subtask delegation policy we introduce the delegation function f_{ij} as follows

$$f_{ij} = \Pr(\text{the node goes to state } (i, j) \text{ after the delegation}). \tag{10}$$

For notational convenience we assume $f_{00} = 0$. $\sum_i \sum_j f_{ij} = 1$, because f_{ij} describes the probability of a complete and disjoint set of events. Indeed, this delegation function also contains the distribution of the number of parallel subtasks in an arriving main task.

$$\Pr(\text{an arriving task is composed by } k \text{ parallel subtasks}) = \sum_{i=0}^{k} f_{i,k-i}. \tag{11}$$

Further more, λ denotes the arrival intensity of a new task. A new task is composed by a given number of parallel subtasks and, similarly to the notations of the previous sections, μ_1 and μ_2 denote the subtask completion rate for local and delegated subtasks, respectively. In the $i \leq 3$, $j \leq 3$ part of the state space the following differential equations describe the evolution of the normalized population vector.

$$\frac{d}{dt} n_{01}(t) = -\mu_1 n_{01}(t) + \mu_2 n_{11}(t) + \mu_1 n_{02}(t) + \lambda f_{01},$$

$$\frac{d}{dt} n_{02}(t) = -\mu_1 n_{02}(t) + \mu_2 n_{12}(t) + \mu_1 n_{03}(t) + \lambda f_{02},$$

$$\frac{d}{dt} n_{03}(t) = -\mu_1 n_{03}(t) + \mu_2 n_{13}(t) + \lambda f_{03},$$

$$\frac{d}{dt} n_{10}(t) = -\mu_2 n_{10}(t) + 2\mu_2 n_{20}(t) + \mu_1 n_{11}(t) + \lambda f_{10},$$

$$\frac{d}{dt} n_{11}(t) = -\mu_2 n_{11}(t) - \mu_1 n_{11}(t) + 2\mu_2 n_{21}(t) + \mu_1 n_{12}(t) + \lambda f_{11},$$

$$\vdots$$

The general form of these equations is

$$\frac{d}{dt}n_{ij}(t) = -(i\mu_2 + \mu_1)n_{ij}(t) + \mathcal{I}(i+1)\mu_2 n_{i+1,j}(t) + \mathcal{I}(j+1)\mu_1 n_{i,j+1}(t) + \lambda f_{ij}, \tag{12}$$

where

$$\mathcal{I}(i) = \begin{cases} 1, & \text{if } i \leq 3, \\ 0, & \text{otherwise.} \end{cases}$$

The number of delegated subtasks from one main task is $\sum_i i \sum_j f_{ij}$, consequently the same number of nodes move to state $(*)$ at the arrival of a main task due to the associated task delegations. The differential equation for the number of nodes in state $(0,0)$ is

$$\frac{d}{dt}n_{00}(t) = -\lambda\left(1 + \sum_i i \sum_j f_{ij}\right) + \mu_1 n_{01}(t) + \mu_2 n_{10}(t) + \mu_2 n_*(t)\ . \tag{13}$$

Finally, the differential equation for state $(*)$ is

$$\frac{d}{dt}n_*(t) = \lambda \sum_i i \sum_j f_{ij} - \mu_2 n_*(t)\ . \tag{14}$$

In this set of differential equations the number of computing units working on a delegated subtask is encoded in two ways

$$n_*(t) = \sum_i i \sum_j n_{ij}(t). \tag{15}$$

From the fact that the normalized population vector sums up to one we further have

$$\sum_i \sum_j n_{ij}(t) + n_*(t) = 1. \tag{16}$$

These two relations can be used to simplify the system description or for sanity check of results obtained from the redundant description.

The state transitions described by the differential equations are illustrated in Figure 5. The vertices represent the possible states of the computing units, the edges represent the state transitions and the associated intensities indicate the transition rates. Note that in some cases intensities depend on some relative population values (which is not the case with a transition graph of a CTMC). Those are the cases when the $\mathbf{k}(\mathbf{n}(t))$ matrix elements depend on $\mathbf{n}(t)$ as it is in (9). This set of differential equations can be solved by numerical procedures, e.g., by Runge–Kutta method.

Starting from a completely idle system, $\mathbf{n}(0) = (1, 0, \cdots, 0)$, Figure 6 depicts the system evolution when $\lambda = 1$, $\mu_1 = 5$, $\mu_2 = 4$ and $f_{21} = 1$. According to Figure 6 the system is not saturated with this load and the normalized population vector converges to a fix point.

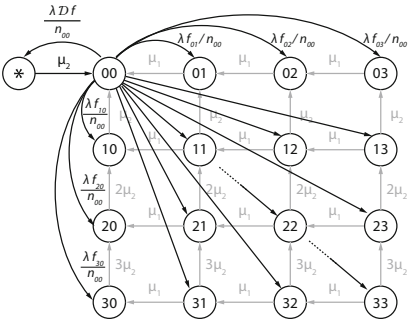

Fig. 5. Transition graph of the mean field model

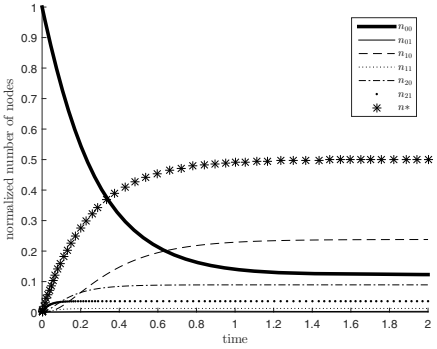

Fig. 6. The results of mean field analysis

The analysis of the behavior of an overloaded system requires an extension of the above set of differential equations with boundary limits to keep the normalized population values between 0 and 1. For example the boundary extension of (13) is

$$\frac{d}{dt}n_{00}(t) = \begin{cases} \bullet, & \text{if } 0 < n_{00}(t) < 1, \\ \max(\bullet, 0), & \text{if } n_{00}(t) = 0, \\ \min(\bullet, 0), & \text{if } n_{00}(t) = 1, \end{cases} \tag{17}$$

where $\bullet$ stands for the expression on the right hand side of (13). The limit of stability is the highest load where $\lim_{t\to\infty} n_{00}(t) > 0$ still holds.

Stability with Autonomous Computing Units

In volunteer computing platforms the project owner has to avoid the system saturation. It is not a trivial problem when the participating computing units are autonomous. For example, if the delegation function is not known because the nodes decide the delegation strategy autonomously the project owner has to find a safe task submission rate, λ, at which the system remains stable independent of the delegation policy of the autonomous computing units.

A safe task submission rate can be obtained by assuming that the autonomous computing units apply always the most inefficient task delegation policy.

The overall resource utilization of the task completion with i delegated and j locally computed subtasks, C_{ij}, can be computed by a recursive relation based on the same considerations as the ones in Section 4.2

$$E(C_{i,j}) = \frac{\mu_1}{\mu_1+i\mu_2}\left(\frac{i+1}{\mu_1+i\mu_2} + E(C_{i,j-1})\right) + \frac{i\mu_2}{\mu_1+i\mu_2}\left(\frac{i+1}{\mu_1+i\mu_2} + E(C_{i-1,j})\right),$$

which accounts for the total resource utilization, since with i delegated and j locally computed subtasks $i+1$ computing units are occupied. Plotting the obtained $E(C_{i,j})$ values similar to Figure 2 and taking the maximal values along

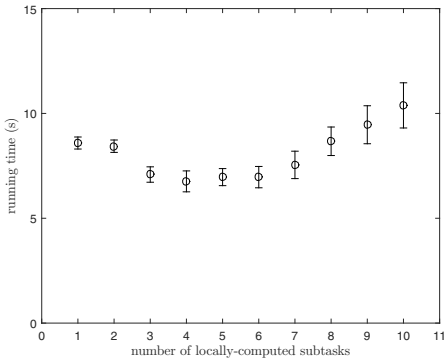

Fig. 7. The results of simulation with 10 subtasks

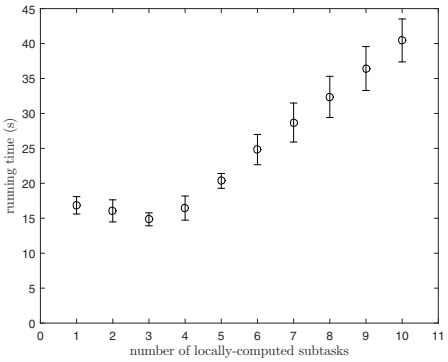

Fig. 8. The results of simulation with 20 subtasks

the $i + j = N$ section planes defines the most inefficient delegation policy. The limit of saturation with this most inefficient delegation policy defines the admissible safe task submission rate of the project owner.

We close the section by mentioning, that the modeling approach applied in this section can be extended for higher levels of hierarchical task subdivisions, but it would complicate the discussion significantly and is out of the scope of this paper.

5 Simulation

To verify the analytical results we have developed an event-driven simulator in `C++` language. The implemented simulation model contains several additional details of a real volunteer computing platform including socket management for data transfer between nodes, task execution in virtual environment, realistic computation delegation strategy with higher level hierarchical subtask division, etc.

5.1 Execution Time Results

Figure 7 shows the execution times of a task with 10 subtasks. In this example, the subtasks can be computed with 10 MFLO (Mega FLoating-point Operations) and the computers have 10 MFLOPS computation capacity in average. The amount of data required for the computation of a subtasks is 3 MB, and the speed of the internet connection is 2 MB/s. Figure 7 shows the average running times and the confidence intervals for 95% confidence level.

The minimum of the task completion time is obtained at 4 locally-computed and 6 delegated subtasks. The running time is 6.76 s in this case.

Figure 8 shows another example with 20 subtasks. It contains the running times up to 10 locally computed subtasks because the linear trend continues

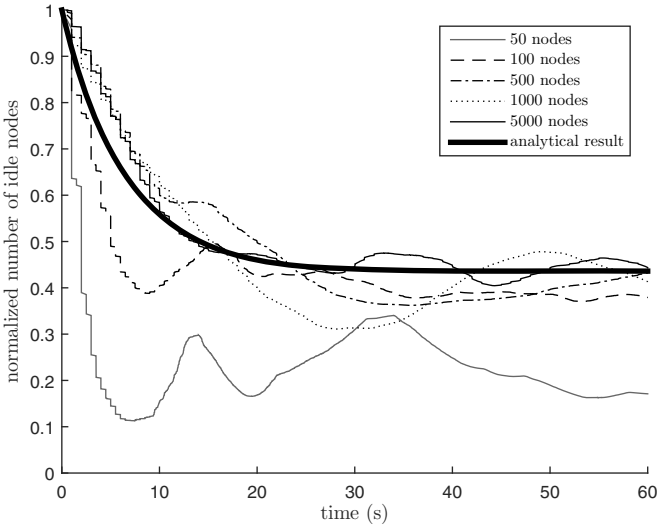

Fig. 9. The results of the simulation and the analytical model for relative number of nodes

over. In this example the subtasks can be computed with 40 MFLO, the other parameters are the same as above. The optimal delegation was found at 3 locally computed subtasks which coincides with the analytical results from Section 4.2. The associated optimal running time is 14.85 s.

5.2 Results of the Overall System Behavior

To verify the results of the mean field analysis in Section 4.3 we collected population results in the simulator. Figure 9 plots normalized number of idle nodes as a function of time for computing platforms with different number of computing units and identical relative loads. In this simulator run tasks were composed by 20 subtasks, the intensity of the task arrivals was $\lambda = 0.005$ 1/s, the calculated task completion rates were $\mu_1 = 0.27$ 1/s and $\mu_2 = 0.18$ 1/s. The figure contains results for computing platforms with 50, 100, 500, 1000 and 5000 nodes as well as the results of the mean field model. Figure 9 supports the intuition that the simulated results converge to the analytical result as the number of nodes increases.

6 Conclusions

The paper presents a performance assessment of volunteer computing platforms. A set of real characteristic features, e.g. task failures, arrivals and departures of computing nodes, restricted availability, local usage, etc. are left for future work. Different modeling paradigms are used for the analysis of performance measures.

References

1. Altman, E., Kameda, H., Hosokawa, Y.: Nash equilibria in load balancing in distributed computer systems. International Game Theory Review **4**, 91–100 (2002)
2. Anderson, D.P.: Public computing: reconnecting people to science. In: Conference on Shared Knowledge and the Web, pp. 17–19 (2003)
3. Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: Fifth IEEE/ACM International Workshop on Grid Computing, pp. 4–10 (November 8, 2004)
4. Anderson, D.P., Fedak, G.: The computational and storage potential of volunteer computing. In: Sixth IEEE International Symposium on Cluster Computing and the Grid, CCGRID 2006, vol. 1, pp. 73–80. IEEE (2006)
5. Anderson, D.P., Walton, R., Fenton, C.: BOINC project. http://boinc.berkeley.edu/
6. Anderson, D.P., Werthimer, D.: SETI@home project. http://setiathome.berkeley.edu/
7. Andrade, N., Cirne, W., Brasileiro, F., Roisenberg, P.: OurGrid: an approach to easily assemble grids with equitable resource sharing. In: Feitelson, D.G., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 61–86. Springer, Heidelberg (2003)
8. Andrade, N., Costa, L., Germóglio, G., Cirne, W.: Peer-to-peer grid computing with the OurGrid community. In: Proceedings of the SBRC, pp. 1–8 (2005)
9. Costa, F., Silva, L., Fedak, G., Kelley, I.: Optimizing the data distribution layer of BOINC with BitTorrent. In: IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, pp. 1–8. IEEE (2008)
10. Farkas, G., Szanto, I., Gora, V., Haller, P.: Extending the BOINC architecture using peer-to-peer application code exchange. In: 2011 Roedunet 10th International Conference (RoEduNet), pp. 1–4. IEEE (2011)
11. Fedak, G., Germain, C., Neri, V., Cappello, F.: Xtremweb: a generic global computing system. In: Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 582–587. IEEE (2001)
12. Garland, M.: Parallel computing with CUDA. In: Proc. of the IEEE Int. Symp. on Parallel and Distributed Processing (IPDPS), Atlanta, GA (April 19–23, 2010)
13. Korpela, E., Werthimer, D., Anderson, D.P., Cobb, J., Lebofsky, M.: SETI@home - massively distributed computing for SETI. Computing in science & engineering **3**(1), 78–83 (2001)
14. Kurtz, T.G.: Strong approximation theorems for density dependent Markov chains. Stochastic Processes and their Applications **6**(3), 223–240 (1978)
15. Luther, A., Buyya, R., Ranjan, R., Venugopal, S.: Alchemi: a. net-based grid computing framework and its integration into global grids. arXiv preprint cs/0402017 (2004)
16. Luther, A., Buyya, R., Ranjan, R., Venugopal, S.: Peer-to-peer grid computing and a. NET-based alchemi framework. High Performance Computing: Paradigm and Infrastructure. Wiley Press, Fall (2004)
17. Nov, O., Anderson, D., Arazy, O.: Volunteer computing: a model of the factors determining contribution to community-based scientific research. In: Proceedings of the 19th international conference on World wide web, pp. 741–750. ACM (2010)
18. Stone, J.E., Gohara, D., Shi, G.: Opencl: A parallel programming standard for heterogeneous computing systems. Computing in science & engineering **12**(3), 66 (2010)
19. Strohmaier, E., Dongarra, J., Simon, H., Meuer, M.: TOP500 project. http://www.top500.org/lists/2014/11/