Research Article Open Access

Yevgeniy Kolokoltsev*, Evgeny Ivashko, and Carlos Gershenson

Improving “tail” computations in a BOINC-based Desktop Grid

https://doi.org/10.1515/eng-2017-0044
Received November 20, 2017; accepted December 8, 2017

Abstract: A regular Desktop Grid bag-of-tasks project can take a lot of time to complete computations. An important part of the process is tail computations: when the number of tasks to perform becomes less than the number of computing nodes. At this stage, a dynamic replication could be used to reduce the time needed to complete computations. In this paper, we propose a mathematical model and a strategy of dynamic replication at the tail stage. The results of the numerical experiments are given.

Keywords: Desktop Grid, scheduling, mathematical model, BOINC.

1 Introduction

Desktop Grid is a high-throughput computing paradigm that provides a cheap, easy to install and support, and potentially powerful computing tool. Desktop Grid is based on a distributed computing system which uses idle time of non-dedicated geographically distributed general-purpose computing nodes (usually, personal computers) connected over the Internet [8]. Desktop Grids popularity is motivated by quick growth of the number of personal computers, and a large increase in their performance, as well as Internet expansion and rise of connection speed.

Volunteer computing is a form of Desktop Grid. The first large volunteer computing project, SETI@home, was launched in 1999, providing the basis for development of the BOINC platform. By now, there are several middleware systems for Desktop Grid computing. However, the open source BOINC platform [6] is nowadays considered as a *de facto* standard among them. Today there are more than 60 active BOINC-based projects using more than 15 million computers worldwide [1]. So, Desktop Grids hold their place among other high-performance systems, such as Computing Grid systems, computing clusters, and supercomputers.

A Desktop Grid consists of a (large) number of computing nodes and a server which distributes tasks among the nodes. The workflow is as follows. A node asks the server for work; the server replies sending one or more tasks to the node. The node performs calculations and when it finishes it, it sends the result (which is a solution of a task or an error report) back to the server. The detailed description of the computational process in BOINC-based Desktop Grids is given in [2].

Because of their nature, task scheduling in Desktop Grids is an important problem. Among other special problems, there is the “tail” computations problem. It is related to a final stage of computations in a Desktop Grid, when the number of tasks is less than the number of computing nodes. Because of the unreliability of the nodes this stage can take much time. Therefore, special mathematical models are needed to reduce duration of the final stage. In this paper a stochastic mathematical model of the “tail” computations based on finishing task probability distribution function is considered.

The structure of the paper is as follows: Section 2 describes the motivation and related works. Section 3 is devoted to a mathematical model of the “tail” computations. Section 4 presents the results of numerical experiments. Finally, Section 5 contains final remarks and conclusions.

***Corresponding Author: Yevgeniy Kolokoltsev:** Department of Physics, Faculty of Sciences, UNAM, Mexico, E-mail: j.kolokoltsev@gmail.com
Evgeny Ivashko: Petrozavodsk State University, Institute of Applied Mathematical Research, Karelian Research Centre of RAS, Russia, E-mail: ivashko@krc.karelia.ru
Carlos Gershenson: Computer Sciences Department of the Institute for Research in Applied Mathematics and Systems (IIMAS) of the National University of Mexico (UNAM), Mexico, E-mail: cgg@unam.mx
Centro de Ciencias de la Complejidad, UNAM, Mexico
SENSEable City Lab, Massachusetts Institute of Technology, USA.
ITMO University, Russian Federation

2 Motivation and Related Works

As it was mentioned in Section 1, BOINC is the most popular Desktop Grid middleware, so we consider scheduling problem based on BOINC workflow.

BOINC is based on the client-server architecture. The client part is software, which is able to employ idle resources of a computer for computations within one or multiple BOINC projects. It is available for computers with various hardware and software characteristics. The server part of BOINC consists of several subsystems responsible for tasks generation, distribution, results verification, assimilation, etc. (see [2]).

BOINC employs several mechanisms to deal with clients' unreliability. One of them is replication. For each computational task, BOINC holds multiple independent copies of the same task which are called *replicas*. The replication level is about 2-5 and is an option of a BOINC project. Replicas are computed independently with different computing nodes, and then their results are compared with the aims of verifying the solution. *Quorum* is the number of equal replicas results needed to verify a task result. This mechanism is used to overcome processing errors and sabotage. Moreover, BOINC settings allow to create and distribute more task replicas dynamically as needed (for example, dynamic replication is used in papers [4, 5, 14] and others).

BOINC employs PULL model to interact with computing nodes. This allows to employ computing nodes that have no direct access to the Internet. But this also leads to impossibility to determine that a node abandons the Desktop Grid and will never return results. So, BOINC uses a *deadline* value which is set for each task instance to limit tasks completion time. If the server does not get a result before the deadline, the task instance is considered lost.

The replication mechanism as a form of redundant computing serves a number of purposes. The main purpose is to increase reliability by increasing the probability to obtain the correct answer in time even if some nodes become unavailable without having finished the task. This helps, in its turn, to improve efficiency (in particular, throughput of successful results). The same time it significantly decreases accessible computing performance. So, the replication value should be as low as possible to provide verified results in time. The problem of search for optimal replication parameters is studied in practice in [12], where a simple mathematical model of tasks computations is presented. Based on this model, the author determines the most suitable replication value using log data of NetMax@home project [13].

A valuable problem of task scheduling in Desktop Grids is optimization of the so called "tail" computations. A distributed computational experiment involving a batch of tasks inherently consists of two stages (see Fig. 1). At the first one the number of tasks is greater than the number of computing nodes (usually it much more in the very beginning). At this stage, available computing power limits the performance, so it is reasonable to supply each node with a unique task (without replication; from the point of view of the makespan replication is useless as it was shown in [7]). With time, the number of unprocessed tasks decreases until it is equal to the number of computing nodes: then the second stage called the "tail" starts. At this stage there is an excess of computing power which could be used to implement redundant computing to reduce overall computing time.

A number of research problems related to specifics of Desktop Grids are connected to the two-staged batch completion. One of them is the fastest batch of tasks completion. In practice, the "tail" computations can take a long time (usually, about two or three times greater than deadline value) because of unreliable hosts. A computational network does not have information on the current status of tasks completion. So, the "tail" can accumulate a lot of nodes that have already abandoned the computing network. As a certain task assigned to such node violates the deadline, it is assigned again to a different node, possibly unreliable too. So, this prolongs the "tail" duration. The solution to the problem is in the redundant computing: currently processing tasks are assigned to vacant computing nodes. This strategy significantly increases the chances that at least one copy is solved in time. Employing this strategy, one should take into account characteristics of computational nodes (availability, reliability, computing power, etc.), processing the same task, accumulated task processing time, expected task completion time, and so on.

This problem is described in [8], where a simple mathematical model is proposed. In this paper we propose more complex mathematical model which provides better results.

The fastest batch of tasks completion problem is more complex if a new batch of tasks should be started immediately after the current batch completion. In this case, redundant computing reduces accessible computing power. One more complicated case is connected to interdependency between the tasks of the new batch and completion of certain tasks in the current batch.

The performance improvements resulting from task replication of batch parallel programs running on a SNOW system of distributed computing is analyzed in [7]. The au-

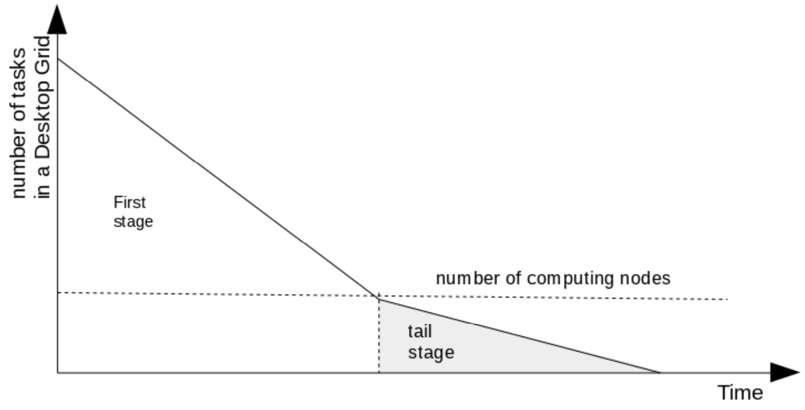


Figure 1: Two stages of batch completion in a Desktop Grid.

thors show that task replication can result in significant speedup improvements, and derived formulas to calculate it. Also, for some workloads when the reliability of a workstation is low, replication can also improve efficiency. Likewise, as job parallelism increases, replication becomes even more beneficial in improving speedup. In the paper, the problem of extra workstations distribution among the tasks with the aim of speedup improvements is also studied. Based on special mathematical models and workload models, it is shown that in case of workstations excess, it is better to increase the number of replicated tasks than to increase the replication level. If there is only one extra workstation, it is better to allocate the extra workstation to the least replicated task. Finally, if there are extra workstations to distribute among two identical programs, distributing the workstations equally between the two programs gives least mean response time for tightly coupled workload and giving all the extra workstations to one of the programs gives least mean response time for loosely coupled workload. Lastly, an analysis of the trade-off between using an extra workstation to increase parallelism or to increase replication is presented. Authors argue that replication can be more beneficial than parallelism for a range of tightly-coupled workloads.

The “tail” computations problem is also studied in [4, 11]. In [11] the following four task replication strategies are proposed and experimentally tested:

- Resource Prioritization — assign tasks to the “best” hosts first.
- Resource Exclusion Using a Fixed Threshold — exclude some hosts and never use them to run application tasks, where filtering can be based on a simple threshold such as hosts clock rates.
- Resource Exclusion via Makespan Prediction — remove hosts that would not complete a task, if assigned to them, before some expected application completion time.
- Task Replication — task failures near the end of the application, and unpredictably slow hosts can cause major delays in application execution. This problem can be remedied by means of replicating tasks on multiple hosts, either to reduce the probability of task failure or to schedule the application on a faster host.

In [4] the following strategy is proposed: When the tail stage starts, all unreliable resources are occupied by instances of different tasks, and the queue is empty. Additional instances are enqueued by a scheduling process: first to the unreliable pool then to the reliable one. This scheduling process is controlled by four user parameters:

- Maximal number of instances sent for each task to the unreliable system from the start of the tail stage.
- Deadline for an instance.
- Timeout, or the minimal time to wait before submitting another instance of the same task.
- Ratio of the reliable and unreliable pool sizes.

3 Mathematical Model

To address the “tail” computations problem we present the following mathematical model. It is a normal practice to assign for each batch of tasks (BoT) a certain number of nodes N registered in the Desktop Grid network. Whereas a volunteer or institutional computational networks are normally heterogeneous, we define by a_k a computing performance of k-th node, where $k = 1, .., N$.

The problem of estimating the computational complexity for each task in a BoT can be challenging in the presence of any iterative computations. However integration or statistical problems are examples where it is possible to know the computational task complexity T in advance. Therefore, for each task in a BoT we define a working time needed for k-th node to finish a task as:

$$T_k = \frac{T}{a_k}, \; k = 1, .., N. \quad (1)$$

Each computing node does not work continuously, so suppose that there is a cumulative probability distribution function (CDF) $F_k(t)$ describing the probability of k-th node to finish a task in certain time. The shape of this CDF and its parameters are discussed in a subsequent section. With $d > T_k$ we define a deadline, that is a time period between task submission to the node, and the moment when the server will reject computation result from that node. Having all thing considered, $F_k(c)$ is probability to successfully finish a task submitted at $t = 0$ to k-th node before time c, and $P_k^{miss} = 1 - F_k(d)$ is probability to miss the deadline.

Redundant computing can be used to reduce the time needed to finish a BoT. At the beginning of the second stage, when a node comes to the server, there is no unique tasks left (see Fig. 1) and $N - 1$ active tasks are already running on the other nodes. If we make a redundant replication of any running task and send its copy to a free node, the probability to finish that task earlier will increase. Formally, this idea can be described with the rule of addition of non-mutually exclusive events.

Suppose, that a probability of task completion before time c on the node l, where it was started at t_l, is given by $F_l(c - t_l)$. If we also assign that same task to the freed node k, at the moment $t_k > t_l$, there will also appear a non-zero probability that a node k will return a result before c, a $F_k(c - t_k)$. Therefore, a probability that *any* of the nodes l or k will return it's result before c is given by:

$$P_{\{k+l\}}(c) = F_l(c-t_l)+F_k(c-t_k)-F_l(c-t_l)\cdot F_k(c-t_k) \geq F_l(c-t_l). \quad (2)$$

At the same time, a probability that both nodes miss their deadlines is given by $P_k^{miss} \cdot P_l^{miss}$. Taking into account that $P^{miss} < 1$ and frequently is selected to be $\ll 1$, the problem of missing deadlines in the "tail" computations can be solved in practice with a few replicas.

We denote with $F_n^T(t)$ a CDF of task return time after it was already submitted to n nodes. If we replicate a task to node k at time t_k, the resulting $F_{n+1}^T(t)$ can be obtained by a following recurrence relation:

$$F_{n+1}^T(t) = F_n^T(t) + F_k(t - t_k) - F_n^T(t) \cdot F_k(t - t_k) \geq F_n^T(t), \quad (3)$$

where $F_0^T = 0$. During the second stage, the task completion CDF $F^{T_i}(t)$ can be stored for each task T_i separately. The maximum number of these functions is N and is reduced continuously as long as tasks complete.

Using the mathematical model described above, one can try to answer the following question: What task is better to replicate in order to reduce "tail" computations? A BoT is finished when the last task of the batch is finished. Therefore, when node k arrives at time t_k, it is reasonable *to equip it with a replica of the task that has the worst probability to be finished before the others*. $F^T(t)$ by definition, is strictly increasing function of t. However, sign$(F^{T_i} - F^{T_j})$ can switch any number of times for any pair of tasks $i \neq j$ on $[t_k, +\infty)$ complicating the identification of the longest predicted task. On the other hand, we know exactly that replication at node k can meaningfully rise F^T for $t \in [t_k, t_k + d]$ only. The necessary task than can be defined as:

$$T_i = \arg\min_{T_i} F^{T_i}(t_k + d), \quad (4)$$

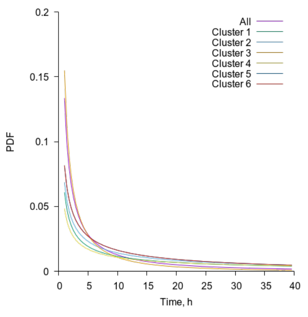
where $F^{T_i}(t_k + d)$ is the probability that task T_i will be finished before the moment $t_k + d$.

Taking into account that the problem defined in Eq. 4 has a discrete nature, various simplifications can be purposed for those cases where a comparison between different F^{T_i} can be derived analytically based on task submission history. The generic case is accessible by a numerical approximation of each F^T that, following our experience, does not require more than 100 reference points and is best done linearly because of oscillations of higher order polynomials at CDF singular points (see below).

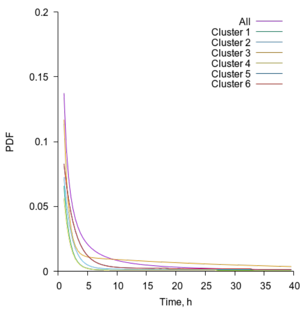
4 Numerical Experiments

We model a client computer activity by a sequence of time periods of availability τ_a and unavailability τ_u. When a node k is available, it dedicates all its CPU power a_k to the current task assigned to it. When a node becomes unavailable, a task computation is immediately suspended and its subsequent resume does not have any additional computational cost. None of internal client machine CPU time scheduling policies are considered.

The work of B. Javadi *et al.* [9] — a statistical analysis of 230,000 host activity tracks of SETI@home project — has showed a presence of 21% of tracks where τ_a and τ_u periods were statistically independent. Activity tracks with the same statistics can be understood as implementations of the same stochastic ergodic process. This process is simple to analyze analytically and permits a random generation of the track statistics without any information loss. To describe such process, it is sufficient to specify the distribution functions for both τ_a and τ_u periods. In [10] the same authors used their previous results, and the discovered ergodic tracks have been additionally classified into six clusters with a subsequent fit of τ_a and τ_u probability densities by well known analytic distributions. The results of this fit are given in Table 4 of the cited work and were used in our track generator. For convenience, we repeat the probability density functions (PDF) for all six clusters in Fig. 2.



a)



b)

Figure 2: Availability (a) and unavailability (b) per-cluster probability density distributions.

In any ergodic process, integration over ensembles can be replaced with integration over time. In our simulations, we use both cases depending on convenience. It should be stressed, however, that some care must be taken in the case of track ensemble generation. When a particular track is used to simulate node activity, the total number of $\tau_a + \tau_u$ periods is typically low, so the statistical correctness of track starting moment cannot be neglected.

First, we calculate analytically the mean available $\bar{\tau}_a$ and mean unavailable $\bar{\tau}_u$ periods. With these values, we define a probability to find a computational node in available state, that is $\bar{\tau}_a/(\bar{\tau}_a + \bar{\tau}_u)$, and use this probability to randomly generate starting state of each new track. Next, we note that track can start at any phase of its first state. A *forward recurrence time* τ_e is defined as the time until the next state change moment. If the process started from available state, τ_e statistics depends on τ_a cumulative distribution F_{τ_a} only (or a τ_u distribution if a node was started from unavailable state). The τ_e CDF is derived in a point process renewal theory, see [3]:

$$F_e(t) = \frac{1}{\bar{\tau}_{a,u}} \int\limits_0^t (1 - F_{\tau_{a,u}}(x))dx \qquad (5)$$

4.1 Task completion time statistics

In the following simulation, we set the same computing power for all the nodes, so that $\alpha_i = \alpha_j = \alpha$ and does not depend on the node's statistical cluster. In this case, we can address a task complexity T using time units (hours), that is equal to the task complexity that can be finished during time T of a continuous computation on any node. For the activity track random generator of each cluster, we continuously send the same task, recording its completion time. A task completion time is defined as a period between its submission moment and a node report with an infinite deadline (the node is always available at these moments). For each cluster, a statistical ensemble consisting of 100,000 completion times was collected. Given these statistical ensembles, an experimental PDF was calculated.

In Fig. 3 we show the PDF bar-plots of a single task completion time for all clusters. Each distribution has a notable peak on its left side corresponding to the case when a node was available from the task start and until its completion T. A theoretical probability of this event is not zero and is given by $1 - F_e(T)$, see Eq. 5. A corresponding CDF of a task completion time F_k is not smooth and has a singularity of the first kind at T, so the corresponding $\sim \delta$ behavior in each PDF is not surprising.

All six clusters have notably different dispersion. However, their shapes are comparable: a starting peak at T followed by a smooth maximum with a long decay tail. In a single cluster, dispersion also depends on the task complexity T, because it defines a mean number of availability periods sufficient to complete a task. By varying T it is possible to observe nearly all variety of typical shapes present on Fig. 3 for any single cluster.

4.2 A BoT computations simulation

In the following simulation, the BoT is composed of a number of equal complexity tasks. Here we consider the case when the total number of tasks is much greater than the number of computing nodes N. It means that a BoT startmoment is far in the past that no correlations related with its submit heterogeneity can be observed at the final stage. We start our simulations with the number of free tasks exactly equal to the number of nodes. At $t_0 = 0$ each node is equipped with a task pending from previous computation and *will complete it successfully* at $t > t_0$ requesting for the new one. The time moment with the system configuration described above, appears in any computation, independently of the deadlines due to the replicate limit

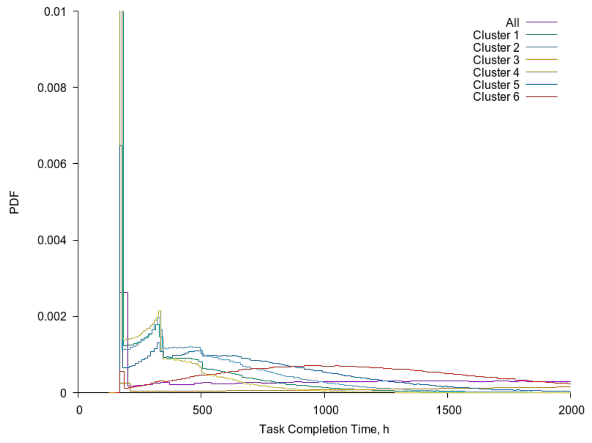


Figure 3: Task completion time probability density distributions for each cluster. The task complexity T =168h (1 week).

in real servers. It is convenient to denote a set of tasks left at t_0 as TL, with $|TL| = N$.

The time period between two subsequent requests for a new task that a node sends to the server is known as the *turnaround time*, τ_{ta}. Turnaround values are random, and in our simplified model they depend on the node activity track statistics, task difficulty T and deadline. In our simulations, τ_{ta} CDF was obtained experimentally for a given set of parameters (the 5-th cluster, T =168h, d = $2T$ =336h). We demonstrate the results based on the 5-th cluster because all the clusters show similar behavior. Unlike the task completion distributions with deadline ignored, τ_{ta} have two singularities: the first one at T and the second one at d. The other features of τ_{ta} statistics are similar to F_k' (Fig. 3).

To generate a time (after t_0) when each node will return to the server to take one of TL tasks, we calculate a forward recurrence time distribution for a turnaround process (same as in Eq. 5) and use a von Neumann random number generator based on that distribution, which is experimental this time.

In Fig. 4 a fraction of nodes $\eta_{TL}(t)$ equipped with unique tasks from TL set simulated for 1000 jobs is shown. As it can be seen, a function $\eta_{TL}(t)$ mean value linearly grows until the second stage t_{II} starts, that is defined as the moment when $\eta_{TL}(t_{II}) = 1$.

The second stage represented in this simulation corresponds to the default BOINC policy and illustrates a BoT completion problem considered in this paper. The default BOINC policy does not have any sense of BoT completeness: when a single task is sent to a computational node, the server will do nothing until either the node response arrives or a deadline comes. Therefore, if the node does not answer until the deadline, a task is left active and will be sent again to the next upcoming node.

From Fig. 4 we observe that $\eta_{TL}(t)$ cut dispersion increases beginning from the first deadline occurrences after t_0. The maximum of dispersion is reached near $t_{II} + d$ moment and a multiply deadlines stretch for a time notably greater than the "best" BoT completion time $(t_{II} + T)$. Observation of the worst case in real computations is unacceptable; in the final subsection we will see how the situation changes when dedicated server policies are used.

4.3 BoT completion time simulations

The BoT completion time t_{end} is the moment when its last task result was reported by a computational node. First, on Fig. 5 we observe simulation results of t_{end} PDF for the standard BOINC policy. Theoretically, if all nodes arrive at the same time at t_0 and do not have unavailability periods, the job completion time is equal to T. Beyond this moment t_{end} PDF is exactly zero. However, as it can be seen on the plot, this case has negligible probability. A notable shortest t_{end} probabilities appear at times nearly twice greater than T. Then, we see the first maximum at nearly 750h. This corresponds to the case when there were no missing deadlines. The second maximum at 1200h correspond to the case where some tasks suffered exactly one deadline. The small subsequent peak corresponds to two deadlines. Presence of the long tail on the right indicates a nonzero probability that even more subsequently deadlines can be missed.

During the second phase, the number of tasks left is reducing continuously. In a standard policy considered above, each node becomes idle by returning a successfully completed task. However if we send a redundant replicates of the running tasks left to these idle nodes, we expect that the probability distribution of t_{job} will change. In the fol-

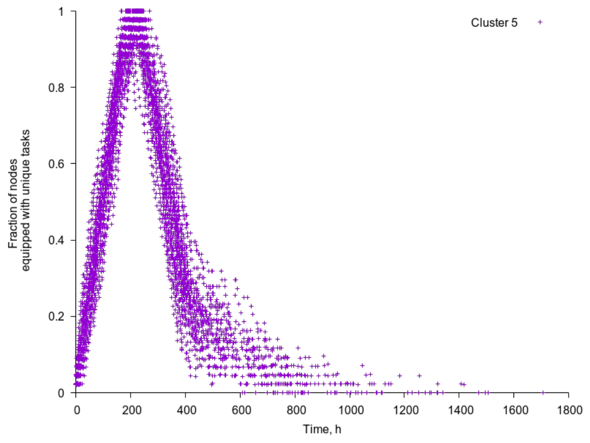


Figure 4: A fraction of nodes $\eta_{TL}(t)$ equipped with unique tasks for default BOINC policy for the 5-th cluster with T =168h and d =336h.

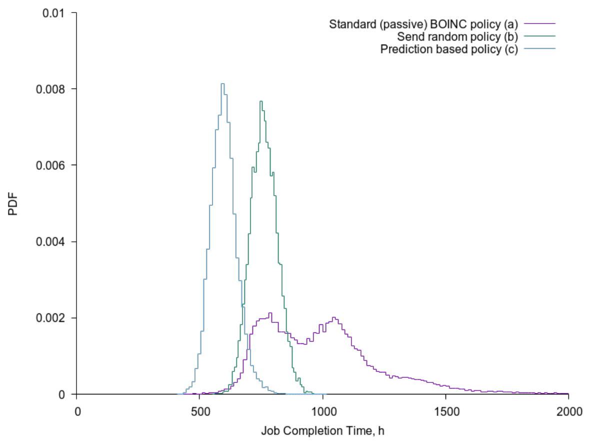


Figure 5: BoT completion time distributions: 5-th cluster, T=168h, d=336h. The standard BOINC (a), random (b) and the Eq. 4 (c) policies are present.

lowing simulation, instead of making a node idle, we send to it a randomly selected task from those that are still not finished. This way a *redundant replication* appears and no nodes are left idle. As a result, we observe an absence of the missing deadline peaks on $t_{job-rand}$ PDF, see Fig. 5. So the deadline problem that brings long tail in t_{job} distribution can be solved simply with a "random" policy. A Gaussian-like distribution obtained, indicates on the good approximation to the central limiting theorem for the sum of availability and unavailability periods for a TL set consisting of only 50 tasks. In the case considered, a mean availability period is less than T so the statistics was sufficient, however a deviations may happen if this criteria is not satisfied.

Random task assignment produces redundant replication for each task without any account that a given task could be already replicated. The policy we consider in this paper is addressed to select such task for each upcoming free node, that has the worst predicted probability to be completed without redundant replication. The simulation results for this case is shown with the curve (c) in Fig 5. In addition to the properties observed for the *random* policy, in this case the mean job completion time is notably reduced.

5 Conclusions and future work

BOINC-based Desktop Grids are widely used to perform long-running scientific computing projects involving thousands of volunteers. BOINC handles huge amount heterogeneous computing tasks and distributed them among thousands of heterogeneous unreliable computing nodes. That is why task scheduling plays crucial role in providing computing performance. The scheduling problem is complicated by the nature of Desktop Grids.

The “tail” computations is one of the important problems in the domain of task scheduling. It arises when the number of tasks become less than the number of computing nodes. Because of unreliability of computing nodes the “tail” stage can take much time despite of excess of available computing power. The special mathematical models-based dynamic replication strategies can be employed to solve this problem.

In this paper we propose a new mathematical model which is based on known in advance probability distribution functions describing the probability of a node to finish a task in certain time. These functions can be constructed based on computing nodes statistics. Using probability distribution functions we estimate the expected task finishing times and propose to equip free node with a replica of the task that has the worst probability to be finished before the others. So, it will reduce the time needed to finish the “tail” stage of computations.

Our numerical experiments based on wide simulations show advances of the approach. Also, there are different criteria which could be used to optimize the “tail” stage of computations. The examples of these criteria are mean computing time reduction, game theoretical multi-objective payoffs, or look-ahead criteria, etc. These are the possible directions of our future work.

Acknowledgement: The authors would like to thank to UNAM’s DGAPA (Dirección General de Asuntos del Personal Académico), that within it’s PREI program made it possible to collaborate with Evgeny Ivashko personally during the 14/04/2015 – 3/07/2015 period. This work is also partially supported by the Russian Foundation for Basic Research (grant numbers 16-07-00622 and 15-29-07974).

References

[1] BOINCstats. In *https://boincstats.com*, 2017-06-22.

[2] David P. Anderson. Boinc: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID ’04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2256-4. 10.1109/GRID.2004.14. URL http://dx.doi.org/10.1109/GRID.2004.14.

[3] Soren Asmussen. *Applied Probability and Queues*. Springer, 2 edition, 2003. ISBN 0-387-00211-1.

[4] Orna Agmon Ben-Yehuda, Assaf Schuster, Artyom Sharov, Mark Silberstein, and Alexandru Iosup. Expert: Pareto-efficient task replication on grids and a cloud. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 167–178. IEEE, 2012.

[5] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic. Smart redundancy for distributed computation. In *2011 31st International Conference on Distributed Computing Systems*, pages 665–676, June 2011. 10.1109/ICDCS.2011.25.

[6] T. Estrada, D.A. Flores, M. Taufer, P.J. Teller, A. Kerstens, and D.P. Anderson. The effectiveness of threshold-based scheduling policies in BOINC projects. In *e-Science and Grid Computing, 2006. e-Science’06. Second IEEE International Conference on*, pages 88–88. IEEE, 2006.

[7] Gaurav D Ghare and Scott T Leutenegger. Improving speedup and response times by replicating parallel programs on a SNOW. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 264–287. Springer, 2004.

[8] Evgeny Ivashko. Mathematical model of a "tail" computation in a desktop grid. In *Proceedings of the XIII International Scientific Conference on Optoelectronic Equipment and Devices in Systems of Pattern Recognition, Image and Symbol Information Processing*, pages 54–59. Southwest State University, Faculty of Fundamental and Applied Informatics, Department of Computer Science, Kursk, Russia, 2017.

[9] Bahman Javadi, Derrick Kondo, J-M Vincent, and David P Anderson. Mining for statistical models of availability in large-scale distributed systems: An empirical study of seti@ home. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS’09. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[10] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, and David P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of seti@home. *IEEE Trans. Parallel Distrib. Syst.*, 22(11):1896–1903, November 2011. ISSN 1045-9219. 10.1109/TPDS.2011.50. URL http://dx.doi.org/10.1109/TPDS.2011.50.

[11] D. Kondo, A.A. Chien, and H. Casanova. Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. 5(4):379–405, oct 2007. ISSN 1570-7873, 1572-9184. 10.1007/s10723-007-9063-y.

[12] I. Kurochkin. Determination of replication parameters in the project of the voluntary distributed computing NetMax@home. In *International scientific conference "High technologies. Business. Society." 14-17.03.2016, Borovets, Bulgaria*, pages 10–12, 2016.

[13] I.I. Kurochkin. The voluntary distributed computing project netmax@home. In *Distributed computing and grid-technologies in science and education: book of abstracts of the 6th international conference*, GRID, pages 233–258, 2009.

[14] J. Sonnek, A. Chandra, and J. Weissman. Adaptive reputation-based scheduling on unreliable distributed infrastructures. *IEEE Transactions on Parallel and Distributed Systems*, 18(11):1551–1564, Nov 2007. ISSN 1045-9219. 10.1109/TPDS.2007.1094.