Replication of "Tail" Computations in a Desktop Grid Project

Evgeny Ivashko[1,2][0000-0001-9194-3976] and Natalia Nikitina[1][0000-0002-0538-2939]✉

[1] Institute of Applied Mathematical Research, Karelian Research Centre of RAS, Petrozavodsk, Russia
{ivashko,nikitina}@krc.karelia.ru,
[2] Petrozavodsk State University, Petrozavodsk, Russia

Abstract. The paper addresses the problem of accelerating the "tail" stage of a computational experiment in a Desktop Grid. We provide the mathematical model of a "tail" stage, describe the setting of simulation experiments and provide their results. Task replication in "tail" phase proves to be efficient in decreasing the duration of "tail" by orders of magnitude.

Keywords: Desktop Grid · Volunteer Computing · BOINC · Replication

1 Introduction

Desktop Grid is a high-throughput computing paradigm which is based on use of idle time of non-dedicated geographically distributed general purpose computing nodes (usually, personal computers) connected over the Internet or by a local access network.

Desktop Grids are an important part of a high-performance computing domain; this concept is used in volunteer computing projects to perform large-scale scientific computations. The most powerful volunteer computing project Folding@home gathers resources exceeding 2 ExaFLOPS[3].

BOINC is an Open Source software platform for Desktop Grid deployment; it is a de-facto standard for volunteer computing. BOINC has a client-server architecture; the server consists of a number of parallel services sharing a database (see [1] for detailed architecture). A client software connects to the server to request tasks, then performs computations and returns the results. The server checks correctness and validates the results, then aggregates them to obtain the solution of the initial problem.

One of the important issues and significant difficulties of a Desktop Grid project implementation is unreliability of clients. A computing node can leave the project after receiving tasks without notification of the server. This leads to losing the tasks. To cope with unreliability, deadline and replication mechanisms are used. Deadline is the expiration date of a task. If the node has not returned

[3] https://stats.foldingathome.org/os

the result before this date, the task comes as lost and is requeued (and to be resent to another node) by the server.

Replication is the form of redundant computing; each task is replicated with a certain factor to be sent to several nodes. The results of computations by several nodes are compared to guarantee the correctness. In addition to correctness, replication is also used to speed up the results of certain tasks receiving having in mind unreliability of computing nodes.

Unreliability is a significant problem at the final stage of a computational experiment, when the number of tasks is less than the number of computing nodes. Then, because of multiple possible deadline violations and requeueing the tasks, the obtaining of the results could be significantly unnecessarily extended. Use of replication at the final stage of an experiment allows to significantly accelerate the experiment. But it is in question what a replication strategy should be used. In this paper we propose a mathematical model of the final stage of an experiment, describe four possible strategies and provide the results of numerical experiments demonstrating replication efficiency.

The structure of the paper is the following. Section 2 presents motivation and related works. In section 3, we describe the mathematical model of the final stage of an experiment. Section 4 gives numerical experiments overview and results. Finally, in section 5, we give the concluding remarks and discuss the results.

2 Motivation and Related Work

The standard computing process of BOINC does not involve dealing with unreliable computing nodes (the ones that never completed the computation of the received tasks and left the Desktop Grid). Instead, the concept of a deadline for computations is used, in case of violation of which the task is considered lost and re-queued for computations. This approach works well in most cases — when one needs to perform an infinite (or very large) number of tasks, mutually independent, of equal value from the point of view of a computational experiment.

However, there are also smaller-scale problems (solved, for example, within the framework of "umbrella" projects or in the Enterprise Desktop Grid), consisting of a sequence of dependent or independent computational experiments, where the results of each individual experiment can be evaluated only when all its tasks are completed. For such problems, the presence of unreliable computational nodes significantly affects the duration of a computational experiment. As the researchers note, the length of the "tail" of the experiment can be several times longer than the final computation period of an individual task (see, for example, [2], [3]). This is due to the fact that the "tail" of the computations accumulates unreliable nodes and the nodes that have left the Desktop Grid. Therefore, in practice, it is important to develop methods for reducing this stage of computations.

Replication is the basic method of reducing the "tail" of computations. Being a form of redundant computing, this mechanism serves several purposes.

Firstly, replication can increase the likelihood of obtaining the correct result in time, even if some nodes become unavailable without completing the computations; secondly, this mechanism allows to increase the efficiency of the system in terms of throughput of correct results. However, redundant computing significantly reduces the available processing power of the Desktop Grid. Therefore, the replication coefficient (the number of copies of a task simultaneously processed on computing nodes) should be as low as possible. The question of finding optimal replication parameters is studied, for example, in [4]: on the basis of a fairly simple mathematical model, the author determines the most suitable parameter values using the work history of the NetMax@home project.

In one of the pioneering works on Desktop Grids [2] (the paper uses the term SNOW — a shared network of workstations), the effect of replication on the speed of task execution is studied. The task is considered completed if it was not interrupted by the owner of the workstation for a certain period of time. If the task is interrupted, it starts firstly on the same or another workstation. Two types of applications are considered: strongly connected (interrupting any task requires a restart of all application tasks) and weakly connected (interrupting a task requires restarting only this task and does not affect the rest). From a mathematical point of view, the joint distribution of N independent identically distributed random variables is analyzed. In this setting, the authors prove that to speed up the application, it is better to make one replica for k tasks than k replicas of a single task. It is also shown that an additional workstation is better be used to replicate a task that has the least number of replicas. In addition, the paper presents an analysis of the trade-off between increased concurrency and replication. It has been shown that in a number of cases, for strongly related tasks, replication is more preferable than parallelization.

A large number of research articles have been devoted to solving the problem of reducing the "tail" of computations. In [3], the following methods for solving this problem are noted:

- redundant computations (replication): several copies of the same task are transferred to different nodes with the expectation that at least one of them will solve it in time;
- scheduling based on reliability and availability assessment: when assigning tasks, preference is given to reliable computing nodes with high availability;
- re-submission: problem tasks are detected and transferred to other nodes for computation.

In [5], the author considers two, previously known, solutions to the problem of reducing the "tail" of computations proposed for the GridBot:

- aggressive replication: replication of tasks in excess of the necessary quorum in case of inactive or too slow computing nodes. However, such an approach, as the author points out, leads to significant losses in computing power and processor time due to redundancy of computations;
- node selection: the use of slow or weakly active computing nodes in the "tail" of computations is inefficient from the point of view of computation time,

therefore, when processing the latest tasks, one needs to select nodes with good characteristics, ideally powerful and continuously available ones.

Along with the two indicated methods, the author of the work offers a new strategy (called "dynamic slack") related to the dynamic setting of the deadline for a task. When transferring the task to the node, the server sets the computation period depending on the completeness of the entire computational experiment and the capacity of the node. At the final stage, a shorter computation period is set, which allows previously inactive nodes to be detected. In this case, it is necessary to maintain a balance, because too short deadline will not allow a reliable node to complete the task in time and will lead to the loss of both computing power and time.

To evaluate the effectiveness of the methods considered in the work, the author conducted a series of experiments. To perform the experiments, the SimGrid simulator was used, in which data on the availability of 100 computing nodes of the SETI@home project were downloaded; three sets of experiments were carried out using various combined strategies for calculating the "tail":

- default strategy without any speedup methods;
- dynamic deadline and node selection;
- dynamic deadline, node selection and aggressive replication.

The results of the experiments showed that the use of a dynamic deadline and selection of a node can significantly reduce the time for calculating the "tail" of a computational experiment, while aggressive replication, on the contrary, leads to a loss of productivity and a delay in computations.

The work [6] addresses the problem of "tail" computation for distributed data processing systems of the MapReduce/Hadoop type. The main difference of such systems from the Desktop Grid in the considered formulation is the ability to remotely interrupt the computation of the task on the node (while, as in the case of Desktop Grid, there is no information about the current state of computations). Also, the work does not limit the number of available nodes, but all replicas start at the same time. It is noted that the following three questions are of interest:

1. What should be the fraction of incomplete tasks for the start of replication?
2. Which number of replicas for each incomplete task to launch?
3. When replicating, whether to cancel the initial incomplete task or not?

The paper considers a strategy with one replication start point in two versions:

- saving the original copy of the incomplete task upon replication;
- aborting the original copy of the incomplete task upon replication.

The main attention is paid to the analysis of the relationship between the completion time of a set of tasks and the expected amount of computations.

Let r be the number of additional replicas of a task. The completion time is determined as follows:

$$E[T] = E[\max_{i \in \{1,2,\ldots,n\}} T_i],$$

where $T_i = \min_{0 \le j \le r} (\tau + X_{ij})$ is the minimal time of completion of either one of the replicas or the initial, ith, task (if it was not aborted), $\tau = 0$ for the initial task (if it was not aborted) or the replication moment. X_{ij} is a random variable from the distribution function of task runtime on the node.

Expected amount of computations is determined as follows:

$$C = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=0}^{r} (T_i - t_{ij}),$$

where n is the number of incomplete tasks.

The author notes that the relationship between the completion time of a set of tasks and the expected amount of computation (cost) depends on two key characteristics of the distribution function of task service time on a node: 1) whether the distribution tail is heavy, light or exponential, and 2) whether the distribution belongs to the type "new faster than old" or "new slower than old".

The authors of work [7], in addition to prioritizing resources and excessive replication, also propose two strategies for excluding nodes from the computation process: based on a simple performance threshold (for example, excluding all nodes with a clock rate less than a given one) and based on a forecast of task completion time (exclusion of nodes that would not complete the task in time if it was assigned to them). In this way, the risk of violating the deadline for computations and delaying computations is reduced. The authors experimentally test the proposed strategies.

From the point of view of resource prioritization, the use of only static information about the clock rate has led to improved application performance, despite the large number of factors affecting the operation of the computing node. This is partly due to the fact that the clock rate correlates with several performance indicators, such as the frequency and time of tasks completion. At the same time, the use of more dynamic information, such as statistics on the availability of a computing node, does not bring the expected benefits due to its weak predictability (complex periodicity, changes in the behavior of the node, etc.). The performance prioritization benefits can be significantly limited when the queue reaches nodes with a relatively low clock speed. Therefore, authors use resource exclusion so that these slow nodes do not delay application execution.

The use of a fixed threshold for clock rate to exclude weak computing nodes has proven itself well in systems with a large dispersion of clock rates. However, in systems with a relatively small dispersion, the exclusion of resources often affected the computational nodes which could be useful in solving the problem. As an alternative to the resource exclusion strategy based on the clock rate, a resource exclusion heuristic was proposed based on the forecasting of the working period, which proved to be better in experiments.

In their work, the authors also explore replication strategies: proactive, reactive, and hybrid ones. Proactive replication aims to replicate all tasks; reactive one — to replicate those tasks whose computation time exceeded the predicted one; finally, the hybrid strategy aims to replicate those tasks for which there is a high probability of exceeding the predicted completion time. The authors use a rather simple prediction of completion time as a ratio of the complexity of the task to the performance of the computing node (the complexity of the task is considered known, and the availability of the node is not taken into account).

The result of the work was the development of a scheduler for the Desktop Grid XtremWeb combining clock prioritization, adaptive exclusion of resources to predict task completion time and reactive replication. The experiments conducted by the authors showed a significant superiority of the developed scheduler over the standard First-Come-First-Served scheduler used in the Desktop Grid.

In this paper we use the same approach as described in [8,9]. We present the more general mathematical model, introduce four heuristic strategies and provide a numerical analysis.

Based on the analysis of research works, we can conclude that the use of intuitive heuristics allows one to get a significant advantage in time compared to the computations without replication. However, a stable time benefit with low redundancy of computations is possible only if the characteristics of computing nodes are taken into account. The main characteristics of the nodes are availability and reliability, defined in this case as follows:

- *reliability* is the ability of a computational node to finish the task before the given time moment (deadline);
- *availability* is the proportion of time periods when the node computes the tasks and the ones when it does not.

The indicated parameters can be expressed by numerical characteristics (see, for example, [10,11]) or, more adequately, with the corresponding statistics — in the form of empirical distribution functions of the task completion time.

Depending on the interrelation of parameters, the following heuristic strategies are possible:

- **Strategy 1** *for available and reliable nodes*: the nodes will compute the tasks in due time, and quickly, therefore, it is necessary to replicate the tasks for which it is possible to reduce the computation time (one by one, starting from expectedly the longest to compute).
- **Strategy 2** *for available but unreliable nodes*: the longer the task has been computing, the more is the probability that the node exited the Desktop Grid, so it is necessary to replicate the longest running tasks.
- **Strategy 3** *for unavailable but reliable nodes*: the longer the task has been computing, the more is the probability that the task will finish soon, therefore, it is necessary to replicate the "earliest" tasks.
- **Strategy 4** *for unavailable and unreliable nodes*: a node may fail to finish the task with high probability, so it is necessary to replicate the tasks with the highest probability to miss a deadline.

Next, we consider the mathematical models of the listed strategies.

3 Mathematical Models of the Experiment Completion

Let us consider Desktop Grid at the moment t_0, when a system consists of n nodes, there is no queue, $n-1$ nodes are computing, and one of the nodes (released or new) requests a task from the server.

We will assume that the computation time of any task is random, independently and equally distributed. Equal distribution means that the distribution function of the task completion time on each node is the same for all tasks (all tasks have the same computational complexity). Independence means that the time for obtaining the result does not depend on the time at which the computations were performed, and the same task is not assigned to the same node.

Suppose that for each node, we know the distribution function of time to complete the computations

$$F_i(x) = F_{i,X}(x) = P(X \le x),\ i = 1, \ldots, n.$$

Then at the moment of time t_0, the expected time to complete the experiment (i.e. finish all the tasks present in the system) is defined as

$$T_{exp} = \max_{i=1,\ldots,n-1} ET_i(t_0).$$

Formally, $ET_i(t_0)$ is written as follows:

$$ET_i(t_0) = \int_{t_0}^{d_i} x dF_i(x|t_0) + \overline{F}_i(d_i|t_0) G(d_i, t_0). \tag{1}$$

Here,

- $F_i(d_i|t_0)$ is the distribution function of time to complete the computations upon condition that they have not finished at the moment t_0.
- $\overline{F}_i(d_i|t_0) = 1 - F_i(d_i|t_0)$ is, at the moment t_0, the probability of the node i to miss a deadline.
- d_i is the due date of computations on the node i (the earlier started the current task, the less)
- $G(d_i, t_0)$ is, calculated at the moment t_0, expected at the moment d_i time to recompute the task again after missing a deadline.

Note that T_{exp}, $T_i(t_0)$, t_0 and d_i are completely different time instances. In its turn, the expected time for recomputing the task again after missing a deadline $G(d_i, t_0)$ consists of the waiting time for the release of the node and the average computation time in case of either completion of the computations in time or the new expected time of recomputing the task again after missing a deadline:

$$G(d_i, t_0) = W(d_i, t_0) + \int_0^d x dF_a(x) + \overline{F}_a(d) \left(d + G\left(t_0 + d_i + W(d_i, t_0) + d\right)\right). \tag{2}$$

Here, the expected time of releasing the node is

$$W(d_i, t_0) = \max\left(d_i, \min_{j=1,\ldots,n;j\neq i} \int_{t_0}^{d_j} x dF_j(x)\right), \tag{3}$$

the index of the completing node is

$$a = \arg \min_{j=1,\ldots,n;j\neq i} \int_{t_0}^{d_j} x dF_j(x) \tag{4}$$

It should be noted that this formula does not take into account the possibility of a new node appearing in the Desktop Grid, as well as the fact that the released node may be busy computing another task for which the deadline has been missed. Furthermore, the calculation of the completion time does not take into account overhead costs, such as the time between the end of the computation and the beginning of the computation of a new task.

Employment in the formulas (1),(2), (3), and (4) of several expected values of various random variables, as well as the above assumptions, lead to the fact that the calculated value of the task completion time on the node i (according to the formula (1)) will significantly differ from the real one. In addition, calculations by the above formulas are quite resource intensive. Therefore, from a practical point of view, it makes sense to move to the averaged values in the formulas (2) and (3). Then

$$G(d_i) = (d_i - t_0) + W' + \int_0^d x dF(x) + \overline{F}(d)G', \tag{5}$$

where

$$G' = d + W' + \overline{F}(d)G' = \frac{d + W' + \int_0^d x dF(x)}{F(d)} \tag{6}$$

and $W' = \frac{1}{2}\int_0^d x dF(x)$.

4 Experimental Results

The experiments were performed on a Desktop Grid simulator program. The simulator is aimed at research of the characteristics of scheduling and replication algorithms in Desktop Grids. It has the form of a simulation model of the process of computational experiment in a Desktop Grid with heterogeneous unreliable computational nodes and heterogeneous by complexity tasks.

The key features of the simulator program are: simulation of the computational process on the BOINC platform; implementation of various probability characteristics of the process; implementation of various heuristics of replication.

The main part of the simulator program is a computational module that simulates tasks distribution over Desktop Grid nodes, collection and processing of results. The varied characteristics of the simulated Desktop Grid are being identified before simulation. The program is implemented in C++.

Empirical distribution functions of task execution time on the nodes and of the task complexities were calculated using statistics of RakeSearch project [12] during its first computational experiment. For the completed tasks, the database field *elapsed_ time* was used to construct an individual distribution function for every individual node.

In order to model unreliability of the nodes, we distinguished the following result outcomes that are set by the client and appeared at least once during the RakeSearch experiment:

- the task was completed successfully;
- the task could not be sent to the client;
- the task started on a node but failed to complete;
- the task was lost (deadline was missed);
- the task was completed, but the result was considered invalid.

The simulation experiments were conducted on a Desktop Grid model with 301 nodes and 300 tasks. The deadline was set to one week, the quorum to one. We considered unreliable nodes with the probability 0.1% of missing a deadline, 0.1% of a computational error or a validation error.

In an experiment, up to 2/3 tasks were replicated. Thus, with 300 initial tasks, up to 500 replicas were generated. All events of missing deadline/computation error/validation error occurred in the 1000 experiments with a proper frequency to evaluate the effect of replication.

In Table 1, we provide the experimental results. The table presents tail length with and without replication, the average lost CPU time on a single node and the total number of lost replicas (i.e. not needed ones), under different replication strategies. The CPU time is considered lost if a node spent it on executing a task already completed by someone else. According to the BOINC mechanisms, we suggested that the server does not notify the nodes about such tasks immediately. Although in BOINC such "never needed" results are rewarded as usual ones to encourage participants, we consider this CPU time as wasted.

Overall, replication in "tail" phase proves to be efficient at a cost of wasted CPU time for computing extra replicas. With the maximal number of replicas $R_{max} = 2$ (i.e. each task is additionally replicated at most once), "tail" length decreases by a factor of 50–100. With further increase of R_{max}, "tail" length and the average lost time do not show significant increase because the extra replicas are actually never generated.

The strategies (1) and (2) perform better in terms of wasted replicas. They experimentally proved to select for the replication such tasks that were highly possible to fail.

Replication strategy	Default tail length	Optimized tail length	Average lost CPU time	Replicas lost in tail
(1)	607 905	9 454 (2%)	3 468	181
(2)	607 974	9 552 (2%)	3 476	181
(3)	608 047	11 972 (2%)	3 527	197
(4)	607 992	11 756 (2%)	3 533	197
(5)	607 984	11 019 (2%)	3 506	194

Table 1. Results of simulations averaged by 1 000 experiments. Considered replication strategies: (1) – replication of a task that would finish the latest; (2) – replication of a task with the largest expected computational time; (3) – replication of a task that started the first; (4) – replication of a task that started the latest; (5) – replication of a random task.

More experiments will follow to determine the best replication strategy for a set of Desktop Grid nodes with given probabilistic characteristics and their subsets basing on analytical investigation of the model and simulation experiments.

5 Conclusion

The problem of "tail" completion is urgent in Desktop Grids due to the unreliability of their nodes. With multiple possible deadline violations and requeueing the tasks, the results can be significantly delayed, negatively impacting the research progress. Replication at the "tail" stage of an experiment allows to speed up its completion by orders of magnitude.

In this paper we address the problem of accelerating the "tail" stage of a computational experiment. We formalize the mathematical model of a final stage of a computational experiment in a Desktop Grid, describe the setting of simulation experiments and provide their results.

Task replication in "tail" phase proves to be efficient in decreasing the duration of "tail" by orders of magnitude. The choice of the particular replication strategy for a certain Desktop Grid requires further investigation, while all considered heuristics demonstrate high efficiency in simulation experiments.

Acknowledgements This work was supported by the Russian Foundation of Basic Research, project 18-07-00628.

References

1. Anderson, D.P.: BOINC: A platform for volunteer computing. Journal of Grid Computing pp. 1–24 (2019)
2. Ghare, G.D., Leutenegger, S.T.: Improving speedup and response times by replicating parallel programs on a SNOW. In: Workshop on Job Scheduling Strategies for Parallel Processing. pp. 264–287. Springer (2004)

3. Kovács, J., Marosi, A.C., Visegrádi, Á., Farkas, Z., Kacsuk, P., Lovas, R.: Boosting gLite with cloud augmented volunteer computing. Future Generation Computer Systems 43, 12–23 (2015)
4. Kurochkin, I.: Determination of replication parameters in the project of the voluntary distributed computing NetMax@ home. Science. Business. Society. 1(2), 10–12 (2016)
5. van Amstel, D.: Scheduling for volunteer computing on BOINC server infrastructures. http://helcaraxan.eu/content/pdf/M2_internship_report_VAN_AMSTEL.pdf (2011)
6. Joshi, G.: Efficient redundancy techniques to reduce delay in Cloud systems. Ph.D. thesis, Massachusetts Institute of Technology (2016)
7. Kondo, D., Chien, A.A., Casanova, H.: Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. Journal of Grid Computing 5(4), 379–405 (2007)
8. Kolokoltsev, Y., Ivashko, E., Gershenson, C.: Improving "tail" computations in a boinc-based desktop grid. Open Engineering 7(1), 371–378 (2017)
9. Ivashko, E.: 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. pp. 54–59 (2017)
10. Essafi, A., Trystram, D., Zaidi, Z.: An efficient algorithm for scheduling jobs in volunteer computing platforms. In: 2014 IEEE International Parallel & Distributed Processing Symposium Workshops. pp. 68–76. IEEE (2014)
11. Miyakoshi, Y., Watanabe, K., Fukushi, M., Nogami, Y.: A job scheduling method based on expected probability of completion of voting in volunteer computing. In: 2014 Second International Symposium on Computing and Networking. pp. 399–405. IEEE (2014)
12. Manzyuk, M., Nikitina, N., Vatutin, E.: Start-up and the results of the volunteer computing project rakesearch. In: Russian Supercomputing Days. pp. 725–734. Springer (2019)