On Sharing Workload in Desktop Grids

Ilya Chernov[0000−0001−7479−9079]

Institute of Applied Mathematical Research, Karelian Research Center of the Russian Academy of Sciences, 185910 Petrozavodsk, Russia chernov@krc.karelia.ru

Abstract. We consider two optimization problems of trade-off between risk of not getting an answer (due to failures or errors) and precision or accuracy in Desktop Grid computing. Quite simple models are general enough to be applicable for optimizing real systems. We support the made assumptions by statistics collected in a Desktop Grid computing project.

Keywords: Optimal work share · Distributed computing · Desktop Grid

1 Introduction

Desktop Grid is a promising paradigm of high-performance computing because of the quick growth both of amount and power of desktop computers, drop of the cost of computing, development of networks (including the Internet). It is a computing system that consists of desktop computers (called computing nodes or just nodes in the sequel) connected by a LAN or the Internet; usually, the nodes are non-dedicated and their power is used by the Desktop Grid only when they are idle [1]. In Volunteer Computing [2] volunteers grant their resources to a project on a free will and can leave at any time; Enterprise Desktop Grids (e.g., [3, 4]) unite resources of an institution and so its structure is more stable.

The classical Desktop Grid architecture is the client-server: the server schedules tasks to nodes and collects the results. In this model, there are no node-to-node connections, even if it is technically possible. Peer-to-peer desktop grids also exist and are studied (e.g., [5,6]); the scheduling problems here are, of course, even more complicated compared to the classical architecture. The challenges of using Desktop Grids for solving calculation-intensive scientific problems include heterogeneity and unpredictable availability of computing nodes, the threat of errors, including sabotage, possible high load of the server, low possibilities for interaction between the nodes.

Heterogeneity [7] means high diversity of computing nodes in terms of both hardware and software type, performance, reliability, availability, predictability, etc. However, the scheduling problem is hard (often NP-hard [8–10]) even under the identical nodes assumption. Nodes grant their idle processor time to desktop grid projects and usually do so without any obligations; therefore they can stop working for the project due to switching to something else, switching off, etc. It is impossible to guarantee the availability of a node at some time and so the structure of the desktop grid is changing in a random way. There are many

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

attempts to predict availability (e.g., [11]) or look for stable patterns (e.g., the review [12]) in availability history. Also, it is important to establish correct deadlines and to use replication so that the total performance increases.

In the classical architecture, the nodes ask the server for tasks, solve them, and report the results to the server. If multiple nodes solve rather quick tasks, the server may fail to process all requests in time and so suffer a DoS attack. This may drastically spoil the performance.

Sabotage and other malicious actions are a problem for volunteer computing, mostly. However, unintentional errors are possible too. Often it is reasonable to sacrifice some performance for double-checking the answers. Besides, sometimes calculations can be done with different precision, e.g., resolution in finite-difference schemes. Higher precision is, naturally, more reliable, but also more time-consuming.

The problems best suited for Desktop Grids are "Bags of Tasks"; a bag of tasks is a set of multiple independent relatively easy computational problems. Much attention has been paid to scheduling bags of tasks in Desktop Grids, optimizing various criteria [8, 12–15]. Often it is reasonable to join simple tasks to complex ones called parcels in order to reduce the server load [16], add test tasks to each parcel [17], mutual check, etc. On the other hand, in some cases a task is itself a parcel of simpler tasks; gathering a statistical sample can serve as an example. The reciprocal of making parcels of tasks is work sharing: tasks of a parcel can be distributed among a few nodes in order to reduce some risk. To our knowledge, there are just a few papers on the subject, e.g., [18].

In this paper, we consider two trade-off problems. The first is the trade-off between sharing work and chances to get the work done. Using fewer nodes to complete a joint task (a parcel) means more nodes for other tasks (parcels), but also higher risk of a switch-off, suspending work, failure, etc. It is obvious that in case of absolutely reliable nodes each is given as large parcel as desired, while in the opposite case of unreliable nodes only single tasks can be scheduled in spite of server load and other drawbacks; otherwise, the expected returns would be too few.

The second problem is the trade-off between reducing the risk by either sharing work (and thus reducing the total performance of the system) or using higher precision (so spending more time on each task). In [19] we propose an optimization criterion called cost per a task. It is the average cost of spent time, with possible replication taken into account, and penalty in case of accepting a wrong answer. The computing model is as follows: tasks are solved independently on identical nodes with some known error probability. Cost of solving a task is known (in time units). Each task is solved until ν identical answers are obtained. This number ν is called the quorum. For binary (yes/no) problems with the unit cost of a task, the spent time is between ν and $2\nu - 1$. If a wrong answer is accepted, some penalty is added to the spent time. An interesting question is not only to determine the optimal quorum given the penalty and error risk, but to estimate the penalty that forces the desired replication. We show that under some reasonable assumptions this dependence is logarithmic and, thus, there is

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

no need to know the penalty values precisely. This is a practically useful result because it is not easy to get precise risk values in terms of the cost of a single task.

Here we develop this idea, expanding from replication to work division. We obtain a similar result (the logarithmic dependence of work share on the penalty threat), which makes the proposed model practically useful.

The nodes are assumed to be identical: so we ignore heterogeneity. One can consider a subset of a large Desktop Grid that consists of similar computers. Also, we ignore unintentional errors and malicious actions; this is reasonable for Enterprise Desktop Grids, where properties of nodes are known, connections are reliable, and saboteurs are unlikely. However, the nodes can be unreliable in the sense that they can suspend doing work for the Desktop Grid project or be switched off at any moment. In section 3 we consider a threat in case of an error, which can be either a wrong answer or a missed interesting result.

2 Work Sharing and the Switch-Off Risk

For the sake of definiteness, let us assume that the tasks collect statistical samples evaluating some function in subsets of a multidimensional set. There are restrictions on the trust level, etc., converted to minimal size N of the sample. So, at least N points must be tested by computing nodes; this sets up N individual tasks that can be grouped into parcels. These tasks can be equally distributed over a few (ν) computing nodes in order to reduce the workload of each node to $n = N/\nu$. Let us call a set of n points to process a *task*.

According to the Desktop Grid paradigm, nodes are not absolutely reliable: they can be switched off, loose connection, return wrong results due to errors or malicious actions, etc. We consider only returning an answer in time: if the answer arrives before the deadline, it is for sure correct.

Probability $q_n = q(t_n)$ of returning an answer depends on the time t_n spent on solving the task; assume that $t = \tau n$ (so τ is the average time to solve an individual task, i.e., to process one point).

Denote $p_n = 1 - q_n$ (the fault probability). The function $p(t)$ is the cumulative distribution function of the time t_f up to the first fault: the node fails if its worktime t is less than the time of work before the fault. The only absolutely continuous distribution with no memory (so that the fault sequence is Markovian) is the exponential distribution; so let us assume that

$$q(t) = \mathrm{e}^{-\frac{t}{T}}, \quad T = E(t_f). \tag{1}$$

Here T is the average time of work up to the first fault.

Failed tasks need to be replicated until the necessary size of the sample is gathered, i.e., until exactly ν nodes return answers.

Statement 1 *The optimal workload for each task and the optimal work sharing are*

$$n = \frac{T}{\tau}, \quad \nu = \frac{N}{n}. \tag{2}$$

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

Proof. The number i of failed tasks can be any, from 0 to infinity; we need exactly ν successful returns, so the distribution is reduced to the negative binomial distribution with parameters ν as the number of successes and q as the success probability in a single try. The negative binomial distribution counts only failures, we need to add all ν successes, so the expectation is

$$E = \sum_0^\infty \binom{\nu+i-1}{\nu-1} q_n^\nu p_n^i \cdot (\nu+i) = \nu + \sum_0^\infty \binom{\nu+i-1}{\nu-1} q_n^\nu p_n^i \cdot i = \nu + \frac{\nu p_n}{q_n} = \frac{\nu}{q_n}. \tag{3}$$

Note that this can be guessed if we consider the scheme as ν geometrical random variables (tries until a success).

Therefore, we need to minimize the function

$$\frac{\nu}{q_n} = \frac{N}{n q_n}, \tag{4}$$

which is equivalent to maximizing $f(n) = n q_n = n \exp(-n\tau/T)$.

Assume that $n \in R$ (not necessarily integer); then the necessary condition for the maximum is $q_n = n q_n \tau/T$, equivalent to

$$n = \frac{T}{\tau}. \tag{5}$$

This can be explained as the number of points processed during the average time before the first failure. If it is less than 1 (highly unreliable nodes), the solution $n = 1$. In case of extremely reliable nodes n can exceed N: in this case, it is reasonable to process more points than the restriction N in order to improve statistical properties of the result.

Statement 2 *The average time spent on gathering the statistics is*

$$\hat{t} = \mathrm{e} \cdot N\tau, \tag{6}$$

i.e., e *time more than time needed to process N points with no failures, independently of the failure probability p (provided that we have enough computing nodes).*

Proof. The average time needed to gather the sample is

$$\frac{\nu}{q_n} \cdot n\tau \tag{7}$$

(the average number of submitted tasks times time per a task). Substitute the optimal values of n and ν to obtain

$$\hat{t} = \mathrm{e} \cdot N\tau. \tag{8}$$

In case of optimal $n < 1$ (unreliable nodes) we need to choose $n = 1$ so that

$$\hat{t} = \mathrm{e}^{\frac{\tau}{T}} \cdot N\tau. \tag{9}$$

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

If, additionally, we are limited in a number of nodes (i.e., $\nu \leq M$), then $n \geq N/M$ and the estimation is even worse. However, nodes can process a few tasks sequentially, which may be better than suffering failures. This case, i.e., the optimal $n = 1$, reduces the parcels to individual tasks. In a highly volatile system, it may be reasonable to collect data point by point using scattered periods of idle processor time. However, here one can face challenges with high server load [16].

Let us estimate the error due to the fact that n is integer. If $T/\tau = n - \varepsilon$, $|\varepsilon| \leq 0.5$, then

$$\Delta\hat{t} = \mathrm{e}^{1+\varepsilon\frac{\tau}{T}} N\tau - \mathrm{e}N\tau. \tag{10}$$

The worst cases are $\approx 107\% N\tau$ (if $\varepsilon = 0.5$, $T/\tau = 1.5$) and $\approx -77\% N\tau$ (if $\varepsilon = -0.5, T/\tau = 1.5$); in case of small τ compared to T the asymptotic estimation is

$$\Delta\hat{t} \approx NT\varepsilon \left(\frac{\tau}{T}\right)^2 = NT\varepsilon \cdot o\left(\frac{\tau}{T}\right). \tag{11}$$

So, in case of reliable nodes rounding is quite safe. If nodes are unreliable, compare two boundary integer values.

To support the assumption about the exponential distribution of the time up to a failure, we considered statistical data gathered from the *RakeSearch*[1] volunteer computing project [20] at the date of 28 March 2018. In fig. 1 we show a histogram of the time before a failure due to any reason was recognized (with a 25 hours deadline). About 70% of values are below 25 hours; we ignore the rest (up to 600 hours) because their density is at most 0.5%

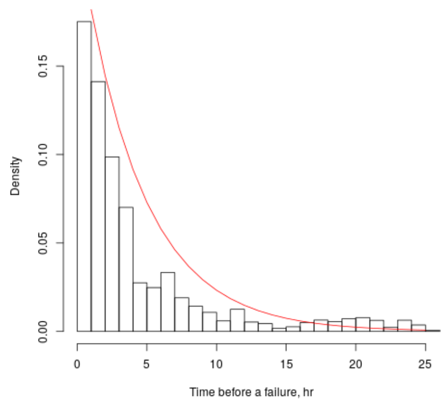

Fig. 1. The histogram of time before failure recognition in the project. The line shows the exponential distribution with $T = \lambda^{-1} = 4.38$ hr.

[1] http://rake.boincfast.ru/rakesearch/

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

3 Precision and Replication

Now let the Desktop Grid search for rare interesting results in a set divided into subsets called *blocks*. The algorithm examines an element and calculates some value; if it is high enough (more than a chosen threshold) the element is interesting and reported to the server.

Often the computational algorithm depends on a parameter that can be called precision: given higher values of this parameter, the algorithm takes more time but is more reliable. This unreliability can have various origins. First, the algorithm itself may produce a wrong answer for some values of its parameters (e.g., if the limit number of terms of a series is insufficient to approximate the sum, or if a descent-type algorithm converges to a false minimum). Second, the task can be to search for an interesting element inside a block; then the number of examined points is the precision parameter and there is always a risk to miss an interesting point unless every point in a block is examined.

Assume that changing some parameter s we are able to reduce the error probability $p(s)$ through more time $C(s) > 0$ spent on solving a task. The functions $C(s)$ and $p(s)$ are continuous. Let $s \geq 1$, $p(1) = p_0 \leq 0.5$, $C(1) = 1$. Again, sharing work between ν identical independent computing nodes is able to reduce the individual load of each node.

Note that this scheme can be used if p is the risk of an error decreased by replication (check by two or more nodes) [19].

The function $p(s)$ is a decreasing function with the lower bound $\bar{p} \geq 0$, which is the unavoidable risk. The cost $C(s)$ increases to infinity as $s \to \infty$.

The probability of finding an interesting answer by at least one computing node is $1 - p(s)^\nu$. If the correct answer is missed, it will be found much later, so that its cost becomes equal to (or even more than) the total cost of the project F. We call F a *threat.* So we need to minimize the function

$$Y(s, \nu) = \nu C(s) + p(s)^\nu F. \tag{12}$$

This is the average cost of an interesting result, with the threat taken into account. Under the made assumptions

$$\lim_{\nu \to \infty} Y = \infty, \quad \lim_{s \to \infty} Y = \infty. \tag{13}$$

So, it is sufficient to consider the optimization problem in a compact set

$$\Omega = \Big\{(s, \nu) : \quad 1 \leq \nu \leq \bar{\nu}, \quad 1 \leq s \leq \hat{s}\Big\}, \tag{14}$$

where $\bar{\nu}$ is high enough and $\hat{s} < \infty$ is such that $C(\hat{s})$ is large enough. As Y is continuous in this compact set, it meets its maximal and minimal values. Denote the interior of Ω by Ω_o.

Statement 3 *If there is a solution $(s^*, \nu^*) \in \Omega_o$, then s^* is a root of the equation*

$$\frac{p'(s^*)}{p(s^*) \ln p(s^*)} = \frac{C'(s^*)}{C(s^*)}, \tag{15}$$

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

and

$$\nu^* = \frac{\ln C(s^*) - \ln F - \ln\ln p^{-1}(s^*)}{\ln p(s^*)}. \tag{16}$$

Note that s^* depend neither on F nor on the optimal ν^*: (15) shows that for the case $s^* > 1$, while for $s^* = 1$ it is obvious.

Proof. If a solution is inside the domain (i.e., in Ω_0), the derivatives with respect to the variables must vanish:

$$\frac{\partial Y}{\partial \nu} = C(s) + Fp(s)^\nu \ln p(s) = 0, \tag{17}$$

$$\frac{\partial Y}{\partial s} = \nu C'(s) + \nu p(s)^{\nu-1} p'(s) F = 0. \tag{18}$$

Divide (18) on (17) to get the convenient condition (15) for the optimal s^* inside the domain.

Also this relation (15) can be rewritten as

$$\frac{d}{ds}\left(\ln \frac{|\ln p|}{C}\right) = 0 \quad \text{or} \quad \frac{d}{ds}\left(\frac{\ln p^{-1}}{C}\right) = 0. \tag{19}$$

So, if the function $\ln p^{-1}(s)/C(s)$ is strictly decreasing, this equation has no roots and, therefore, the only solution is $s^* = 1$: the cost is growing too quickly with respect to the precision. Note that this function can not be strictly increasing because for large s the function $p(s)$ needs to decrease slowly (tending to $\bar{p}$). This gives us a no-root condition:

Statement 4 *The optimal $s^* = 1$ if*

$$G(s) = \frac{\ln p^{-1}(s)}{C(s)} \tag{20}$$

is strictly decreasing.

such conditions make exploiting high precision is useless and therefore are practically convenient. Let us derive other sufficient no-root conditions for this equation.

Statement 5 *If*

$$p(s) \geq p_0^{C(s)}, \tag{21}$$

then the optimal $s^* = 1$.

Proof. Consider the differential inequality

$$\frac{p'(s)}{p(s)\ln p(s)} \leq \frac{C'(s)}{C(s)}. \tag{22}$$

It can be transformed to $d \ln|\ln p| \leq d \ln C$ and integrated from 1 to any C:

$$\ln \frac{\ln p}{\ln p_0} \leq \ln C \quad \text{or} \quad \frac{\ln p}{\ln p_0} \leq C. \tag{23}$$

Finally, we get (23).

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

Note that inequality (23) holds for large s if $\bar{p} > 0$, so solutions, if any, can exist only for reasonably small costs s. Also note that the inequality can be an identity for a special class of functions

$$\tilde{p}(s) = p_0^{C(s)}. \tag{24}$$

In this case the unavoidable risk $\bar{p} = 0$, necessarily. Here lies an important case of $C = s$, $p = 2^{-s}$: linear performance drop with exponential increase of accuracy. For such risk $p(s)$ the cost Y depends, actually, on the joint variable $x = s\nu$. Then either refuse to share work ($\nu^* = 1$) or choose he minimal precision ($s^* = 1$); then, using (16), get

$$x^* = s^*\nu^* = \frac{\ln F + \ln\ln 2}{\ln 2}. \tag{25}$$

In both cases the optimal solution logarithmically depends on the threat F.

Excluding the cases when this equation is an identity and when it has no roots, we can obtain all possible values of s^* which depend only on the properties of the computing system: functions $p(s)$ and $C(s)$.

The right-hand side of equation (16) should be positive; otherwise the optimal $\nu^* = 1$ (the boundary of the domain Ω). So, we see, that the dependence of ν^* on F is logarithmic, i.e., one need not know F precisely: quite coarse estimations are sufficient. In the same way, ν^* is insensible with respect to the cost $C(s)$.

Finally, let us establish the uniqueness in the following sense:

Statement 6 *Let s be given; the optimal ν^* inside Ω_0 is at most one.*

Proof. Consider equation (17). Its left-hand side is an increasing function of ν (remember that $p < 1$ and thus $\ln p < 0$). So, it is not able to vanish more than once.

4 Conclusion

In this paper, we consider rather simple yet general mathematical models of work sharing in a computing system of Desktop Grid type. We consider the trade-off between the risk of not getting an answer from some of a few nodes doing a long task and the risk of not getting an answer from some of many nodes doing short tasks. The average time is shown to be constant in case of exponential distribution of availability intervals. Formulae for any distribution are also available. Then we consider a threat of additional cost for missing a valuable result and the trade-off between careful (high-precision) expensive search by a few nodes and cheaper (low-precision) search by multiple nodes under the assumption of independence of the nodes. The main results are the logarithmic dependence of the optimal solution on the penalty values (so that an order of magnitude is sufficient to be known) and the threshold decrease rate of the risk as a function of cost. Unless the risk is exponentially decreasing, using the lowest precision with cross-checking by many nodes looks optimal.

Суперкомпьютерные дни в России 2018 // Russian Supercomputing Days 2018 // RussianSCDays.org

Acknowledgements

The research was supported by the Russian Foundation of Basic Research, projects 18-07-00628, 16-07-00622.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. The International Journal of High Performance Computing Applications. **15**(3). 200–222. (2001).
2. Sarmenta, L.F., Hirano, S.: Bayanihan: Building and studying web-based volunteer computing systems using Java. Future Generation Computer Systems. **15**(5). 675–686. (1999).
3. Kondo, D., Chien, A., Casanova, H.: Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. Journal of Grid Computing. **5**(4). 379–405. (2007). doi: 10.1007/s10723-007-9063-y
4. Ivashko, E.: Enterprise desktop grids. In: BOINC-based High Performance Computing: Fundamental Research and Development. Proceedings of the Second International Conference BOINC:FAST2015, pp. 16–21. Petrozavodsk, Russia (2015).
5. Kwan, S., Muppala, J.: Bag-of-tasks applications scheduling on volunteer desktop grids with adaptive information dissemination. In: Local Computer Networks (LCN), 2010 IEEE 35th Conference on, pp. 544–551. IEEE (2010).
6. Rius, J., Cores, F., Solsona, F.: Cooperative scheduling mechanism for large-scale peer-to-peer computing systems. Journal of Network and Computer Applications. **36**(6). 1620 – 1631. (2013). doi: http://dx.doi.org/10.1016/j.jnca.2013.01.002
7. Anderson, D., Reed, K.: Celebrating diversity in volunteer computing. In: System Sciences, 42nd Hawaii International Conference on, pp. 1–8. IEEE (2009).
8. Xhafa, F., Abraham, A.: Computational models and heuristic methods for grid scheduling problems. Future Generation Computer Systems. **26**(4). 608–621.(2010). doi: 10.1016/j.future.2009.11.005
9. Casanova, H., Dufossé, F., Robert, Y., Vivien, F.: Scheduling parallel iterative applications on volatile resources. In: 2011 IEEE International Parallel & Distributed Processing Symposium, pp. 1012–1023. IEEE (2011). doi: 10.1109/IPDPS.2011.97
10. Chmaj, G., Walkowiak, K., Tarnawski, M., Kucharzak, M.: Heuristic algorithms for optimization of task allocation and result distribution in peer-to-peer computing systems. International Journal of Applied Mathematics and Computer Science. **22**(3). 733–748. (2012).
11. Kianpisheh, S., Kargahi, M., Charkari, N.M.: Resource availability prediction in distributed systems: An approach for modeling non-stationary transition probabilities. IEEE Transactions on Parallel and Distributed Systems. **28**(8). 2357–2372. (2018). doi: 10.1109/TPDS.2017.2659746
12. Durrani, N., Shamsi, J.: Volunteer computing: requirements, challenges, and solutions. Journal of Network and Computer Applications. **39**. 369–380. (2014). doi: 10.1016/j.jnca.2013.07.006
13. Choi, S., Kim, H., Byun, E., Hwan, C.: A taxonomy of desktop grid systems focusing on scheduling. Technical report KU-CSE-2006-1120-02, Department of Computer Science and Engeering, Korea University (2006).
14. Estrada, T., Taufer, M.: Challenges in designing scheduling policies in volunteer computing. In: Cérin, C., Fedak, G. (eds.) Desktop Grid Computing, pp. 167–190. CRC Press. (2012).

15. Khan, M., Mahmood, T., Hyder, S.: Scheduling in desktop grid systems: Theoretical evaluation of policies and frameworks. International Journal of Advanced Computer Science and Applications. **8**(1). 119–127. (2017).
16. Mazalov, V., Nikitina, N., Ivashko, E.: Task scheduling in a desktop grid to minimize the server load. In: Malyshkin, V. (ed.) Parallel Computing Technologies, International Conference on, vol. 9251, pp. 273–278. Springer. (2015).
17. Yu, J., Wang, X., Luo, Y.: Deceptive detection and security reinforcement in grid computing. In: 5th International Conference on Intelligent Networking and Collaborative Systems, pp. 146–152. (2013).
18. Bazinet, A., Cummings, M.: Subdividing long-running, variable-length analyses into short, fixed-length BOINC workunits. Journal of Grid Computing. **14**. 429–441. (2016). doi: 10.1007/s10723-015-9348-5.
19. Chernov, I., Nikitina, N.: Virtual screening in a desktop grid: Replication and the optimal quorum. In: Malyshkin, V. (ed.) Parallel Computing Technologies, International Conference on, vol. 9251, pp. 258–267. Springer (2015).
20. Manzyuk, M., Nikitina, N., Vatutin, E.: Employment of distributed computing to search and explore orthogonal diagonal latin squares of rank 9. In: Proceedings of the XI All-Russian research and practice conference ”Digital techologies in education, science, society”, pp. 97–100. Petrozavodsk, Russia. (2017).