Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 475–480
ISBN 978-83-60810-14-9
ISSN 1896-7094

On the Robustness of the Soft State for Task Scheduling in Large-scale Distributed Computing Environment

Harumasa Tada
Faculty of Education
Kyoto University of Education
1, Fujinomori-cho, Fukakusa, Fushimi-ku,
Kyoto 612-8522, Japan
Email: htada@kyokyo-u.ac.jp

Makoto Imase, Masayuki Murata
Graduate School of Information Science
and Technology
Osaka University
1-5, Yamadaoka, Suita, Osaka 565-0871, Japan
Email: {imase,murata}@ist.osaka-u.ac.jp

***Abstract*—In this paper, we consider task scheduling in distributed computing. In distributed computing, it is possible that tasks fail, and it is difficult to get accurate information about hosts and tasks. WQR (workqueue with replication), which was proposed by Cirne et al., is a good algorithm because it achieves a short job-completion time without requiring any information about hosts and tasks. However, in order to use WQR for distributed computing, we need to resolve some issues on task failure detection and task cancellation. For this purpose, we examine two approaches—the conventional task timeout method and the soft state method. Simulation results showed that the soft state method is more robust than the task timeout method.**

I. Introduction

DISTRIBUTED computing, in which large-scale computing is performed using the idle CPU times of many PCs, has attracted considerable attention recently. The jobs executed in distributed computing comprise many tasks. These tasks are allocated to PCs and are processed in parallel. Some well-known active distributed computing projects are SETI@home[1] and distributed.net[2].

So far, distributed computing has been mainly used in the limited area of scientific computing for analysis of protein folding, climate simulations, nuclear physics, etc. In these type of applications, the jobs are usually so large that they often take several months to be completed. For such large jobs, the effect of task scheduling on the job completion time is relatively small. Therefore, an efficient task scheduling algorithm is not that important in traditional distributed computing projects.

In the future, it is expected that distributed computing will be applied to various types of large-scale computing applications such as analysis of DNA, data mining, simulations of atmospheric circulation or ocean circulation, structural and stress analysis and fluid analysis of the air resistance of cars or planes and the water resistance of ships. These type of applications require many medium-sized jobs that take several hours or days to complete. The job completion time of such medium-sized jobs are affected by task scheduling.

In this paper, we discuss task scheduling in distributed computing.

Task scheduling in distributed computing has two main goals. The first goal is to minimize the job completion time. In order to complete a job, all tasks that are executed on various hosts should be completed. The delay in a single task can affect the completion time of the whole job. Therefore, the scheduler has to monitor the task execution at each host and perform appropriate actions in response to the change in a situation. The second goal is to minimize the wastage of CPU cycles. In distributed computing, it is common to replicate tasks to achieve good performance. However, using task replicas results in a wastage of CPU cycles. In traditional distributed computing projects, this wastage of CPU cycles has been tolerated because such CPU cycles otherwise go into idle cycles. However, the wastage of CPU cycles implies the wastage of electric power if PCs have power-saving function. Considering recent trend of energy saving, the importance of the reduction in wastage of CPU cycles in distributed computing is increasing.

Task scheduling is not a new problem. It has been studied in the area of parallel computers or clusters. However, the distributed computing systems targeted in this paper are different from parallel computers or clusters. The characteristics of distributed computing are listed as follows:

- Hosts are heterogeneous and autonomous[3].
- It is difficult to obtain good information about the hosts[4].
- Hosts are often behind NATs (network-address translations) or firewalls[5].
- Hosts are frequently turned off by users[5].

In BOINC[5], the well-known distributed computing platform, task scheduling is performed on the basis of the information regarding the processing power of each host and the estimation of the processing time of each task. When this information does not reflect the actual performance, the job completion time deteriorates.

In this paper, we investigate a task scheduling method known as WQR (workqueue with replication)[6], which was

originally proposed for heterogeneous Grid environments. The main feature of WQR is that it does not require any kind of information about the hosts or tasks. WQR has the same performance as the existing scheduling method used, which requires informations on the hosts and tasks. For this reason, the use of the WQR method in distributed computing appears promising. The drawback of WQR is the wastage of CPU cycles. In WQR, some CPU cycles are wasted because tasks are replicated and processed by multiple hosts.

There are some issues to consider when WQR is applied to distributed computing. First, the original WQR method does not take into account task failures and therefore it is possible that the job will not be completed if task failures occur frequently. In distributed computing, however, task failures are not exceptional because the hosts are frequently turned off by users. Second, the original WQR method assumes that the scheduler is able to cancel executing tasks by sending messages to the hosts. However, in distributed computing, it is difficult for the scheduler to send messages to the hosts because of NATs or firewalls. Therefore, in order to use the WQR method in distributed computing, it is necessary to add additional mechanisms to enable task failure detection and task cancellation.

A common approach to solve these issues is to set a timeout for task execution. Most existing distributed computing platforms use this approach[7]. If a scheduler does not receive the result of an allocated task before the timeout, the scheduler creates another replica of the task and allocates it to another host. We call this method as the task timeout method. The task timeout method ensures the completion of all tasks of the job. The problem in this approach is the determination of the appropriate timeout value. The appropriate timeout value depends on the average task execution time and therefore cannot be determined uniquely. The wrong timeout value may cause a delay in job completion and wastage of CPU cycles.

Another approach involves the use of the soft state protocol, which improves the robustness of distributed systems[8]. In this approach, hosts and schedulers exchange messages periodically in order to monitor each other's states. Messages are sent in a best-effort (unreliable) manner[9], that is, they are not retransmitted if they are lost.

In this study, we used the task timeout method and the soft state method with WQR and compared their performances through simulations. The simulation results showed that the soft state method had a better performance than the task timeout method.

The rest of this paper is organized as follows. In section II, we define the distributed computing model and the terms used in this study. In section III, we provide an overview of WQR. In section IV, we explain the issues in applying WQR to distributed computing. In section V, we describe two approaches to deal with these issues. In section VI, we evaluate the performance of these approaches through simulations. Finally, in section VII, we conclude the paper.

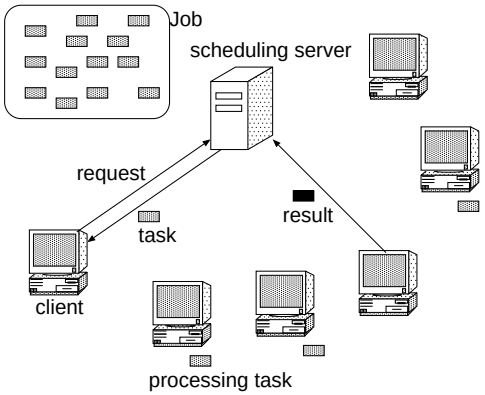

Fig. 1. Model of the distributed computing environment

II. The Distributed Computing Environment Model

Distributed computing involves performing large-scale computations using the idle CPU cycles of many PCs in a network. The participating PCs are known as *hosts*. We use the term *users* to refer to the people using the hosts. A *task* is a unit of scheduling that consists of a program and input data. A *job* is defined as a set of tasks.

Task scheduling involves the allocation of tasks to hosts along with the allocation of the order of execution. Like most distributed computing projects, we consider Bag-of-Tasks applications in which tasks are completely independent[4].

Fig. 1 shows the model of the distributed computing environment considered in this paper. It consists of one scheduler and many hosts.

When the scheduler receives a job, it allocates the tasks in the job to hosts. The hosts receive the tasks from the scheduler and send back the results after task completion. We assume that the transfer of tasks and results are performed in a reliable manner using retransmissions.

However, hosts sometimes crash because of hardware failures or shutdowns by users. When hosts crash, their tasks also fail. Task failures are not notified explicitly to the scheduler.

In this paper, we distinguish between task failure and task cancellation. The former implies that task execution is stopped unexpectedly due to crashing of the host. The latter implies that the task execution is stopped as the task is considered unnecessary by the scheduler.

III. Workqueue with Replication

Workqueue with replication (WQR)[6] is a task scheduling algorithm originally proposed for large-scale distributed systems such as Grids. The main feature of WQR is that it does not require any kind of information about the hosts or tasks. The performance of WQR is equivalent to that of FPLTF[10] or Sufferage[11], which require information about the hosts and tasks[6].

The WQR algorithm uses task replication to achieve a good performance. The scheduler chooses tasks in an arbitrary order and sends them to the hosts. When a host completes its task, it sends the result back to the scheduler and receives a new task

from the scheduler. This scheme continues until all the tasks in the job are allocated. In the simple Workqueue algorithm, hosts that complete their tasks become idle. In WQR, however, these hosts are allocated replicas of tasks that are still running. When a task replica finishes at any host, its other replicas are cancelled.

Using task replication, WQR can improve the performance of a distributed computing by reducing the delay in completion of tasks allocated to slow/busy hosts. Task replication increases the possibility that at least one replica is allocated to a fast/idle host.

However, to avoid wastage of CPU cycles, there is a predefined limit on the number of task replicas. Tasks are replicated until the number of replicas reaches this predefined limit.

IV. Issues in application of WQR to distributed computing

The concept of WQR is very desirable in a distributed computing environment in which it is difficult to obtain accurate information about the performance of hosts. However, the original WQR algorithm is not recommended for use in a distributed computing environment. When the WQR method is applied to distributed computing, there are some issues to consider.

A. Detection of task failure

When a task fails, the scheduler should detect the failure and reschedule the failed task to a different host. However, it is very difficult for the scheduler to detect task failures without any notification.

In the original WQR method, the number of task replicas that can be created is limited. If all the replicas of a task fail and these failures are not detected by the scheduler, the task can never get completed. Such a situation can be avoided if the number of task replicas is unlimited. However, this increases the wastage of CPU cycles[6].

B. Explicit task cancellation

In the original WQR method, when a task replica is completed by any host, the executions of its other replicas are cancelled in order to reduce the wastage of CPU cycles.

However, in distributed computing, it is difficult for the scheduler to cancel the tasks executing on hosts. This is because in a distributed computing environment, the hosts are generally behind NATs or firewalls. In this situation, communications are always initiated by the hosts and not by the scheduler. The scheduler cannot send any messages unless the hosts open a connection to the scheduler. Therefore, even though the scheduler receives the results of the completed task, it cannot send messages to the other hosts to cancel the replicas.

V. Extensions of WQR

We consider two approaches to improve the WQR method, i.e., the task timeout method and the soft state method, in order to deal with the issues discussed in section IV.

A. Task timeout method

In BOINC, the task timeout method, which has an upper limit for task execution time, is used.

The scheduler allocates a task to a host and sets the timeout value on the timer. If the scheduler does not receive the result of the task from the host before the timeout period elapses, it reschedules the task to another host. This method ensures the completion of all tasks of the job.

A problem with the timeout method is the determination of an appropriate timeout value. The appropriate timeout value depends on the average task execution time. An inappropriate timeout value delays job completion and wastes many CPU cycles. For example, if the timeout value is too small, the scheduler creates redundant replicas of tasks that are still running, while if the timeout value is too large, there is a delay in the detection of failed tasks by the scheduler.

The explicit task cancellation mentioned in section IV-B is impossible in the task timeout method. All task replicas are executed until they have completed or failed.

B. Soft state method

The soft state method was originally proposed for state management of communication protocols[12]. It is characterized by periodic refreshing of information and the initialization of information by timeout. Lui et al. showed that the use of the soft state method makes communication protocols highly robust[8].

In the soft state method, the scheduler and hosts exchange information with each other. Hosts that are executing tasks send refresh messages to the scheduler periodically. On receiving the refresh message, the scheduler sends a reply message to the host. Messages are sent in a best-effort (unreliable) manner, that is, they are not retransmitted if they are lost. If the scheduler does not receive a refresh message before the message timeout, it reschedules the corresponding task to another host. On the other hand, if the host does not receive a reply message before the message timeout, it aborts its task and sends a request for a new task to the scheduler.

The main feature of this method is that the scheduler and the hosts are aware of each other's states due to the periodic exchange of messages. On receiving refresh messages from the hosts, the scheduler confirms that the allocated tasks are running normally. The reply messages from the scheduler to the hosts confirm that the scheduler is still waiting for the results of the tasks, that is, the tasks are not cancelled. Thus, the soft state method can perform explicit task cancellation, which cannot be achieved with the task timeout method. By stopping the flow of refresh messages, the scheduler can inform the host that the task is no longer necessary.

However, problems can arise if message losses occur frequently. If refresh messages are lost consecutively, the scheduler regards the corresponding task as failed and creates redundant replicas of the task. If reply messages are lost consecutively, the host regards its task replica as cancelled and aborts the necessary task replica. This false task abortion is another cause of task failure.

In order to exchange messages with the scheduler, hosts should always be connected to the network. However, since the sizes of refresh and reply messages are very small, the network load added by these messages is negligible compared to the tasks and their results.

VI. Performance Evaluation

We evaluated the performance of the two methods mentioned in section V through simulations.

We assumed that the network transfer times were negligible because our target applications were CPU bound. As the performance criteria, we considered the job completion time and the number of wasted CPU cycles The job completion time is the time between the start of the first task and the completion of the last task. The wasted CPU cycles are the sum of all the CPU cycles that are used for executing task replicas that did not contribute to the final result. Such task replicas include cancelled replicas, failed ones, and redundantly completed ones.

In the rest of this section, we will refer to the task timeout method and the soft state method as the TT and SS methods, respectively.

A. Simulation settings

In our simulations, the scheduler runs a single job. Therefore, all hosts execute tasks of the same job. Further, each host executes only one task at a time.

The processing power of each host is taken from a uniform distribution $U(1, 7)$ and therefore the average processing power of the hosts is 4. The total processing power of all the hosts in the system is fixed to 1000.

The size of a task is defined as the processing power required to process the task. For example, a task with a size of 10 is processed in 5 s by a host with a processing power of 2. The average task size (denoted as μ) is 10000 (large tasks) or 100 (small tasks). The size of each task is an integer taken from a normal distribution with a standard deviation of 0.4μ. In both cases, the total size of all tasks included in a job is fixed as 1000000.

The number of task replicas is limited to 4 on the basis of the results of the study by Cirne et al.[6], which states that the performance of WQR with a limit of 4 is close to that when the number of task replicas is unlimited.

The task failure rate is defined as the inverse of the MTBF (mean time between failures) of tasks. All tasks have the same failure rate regardless of their size. This reflects the fact that large tasks stay in the hosts for a long time and therefore they are more likely to be involved in host crashes.

The message loss rate is defined as the ratio of lost messages to total messages sent. If the message loss rate is 0.1, it implies that 10% of the messages are lost.

For the TT method, the simulations were performed by changing the task timeout value. We used three timeout values e_s, $2e_s$, and $4e_s$. e_s is the estimated average task execution time given by $e_s = \mu/p$, while p is the average processing power ($p = 4$, as mentioned in section VI-A). For example, if the average task size is 10000, then the task timeout values are 2500, 5000, and 10000. In the following graphs, "TT-X" indicates a task timeout method with timeout value X.

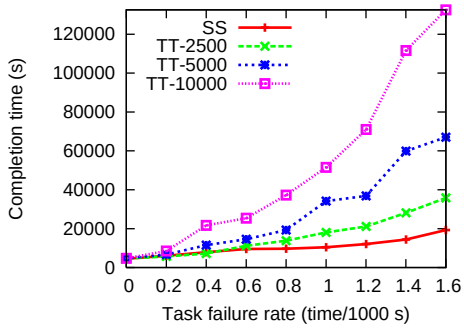

Fig. 2. Plot of the job completion time ($\mu = 10000$)

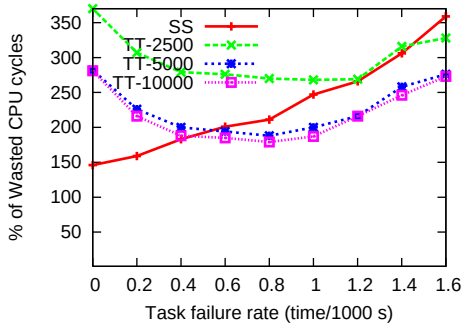

Fig. 3. Plot of the wasted CPU cycles ($\mu = 10000$)

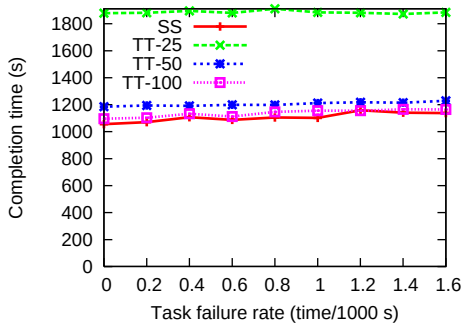

Fig. 4. Plot of the job completion time ($\mu = 100$)

In the simulations of the SS method, refresh messages were sent every 10 s and the message timeout value was set to 100(s). This implies that 10 or more consecutive message losses resulted in rescheduling or abortion of tasks, as mentioned in section V-B.

B. Simulation results

For each parameter setting, we performed 10 simulations using different jobs. The presented results are the average of the results of the 10 simulations.

1) Effect of task failures: The task failure rate was changed for all simulations. The message loss rate was fixed at 0.

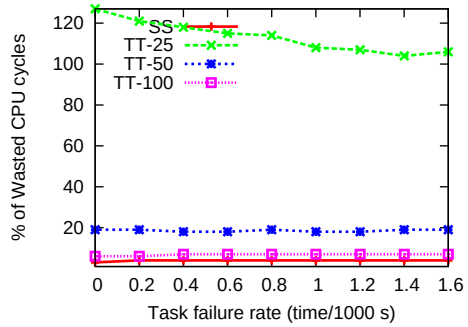

Fig. 5. Plot of the wasted CPU cycles ($\mu = 100$)

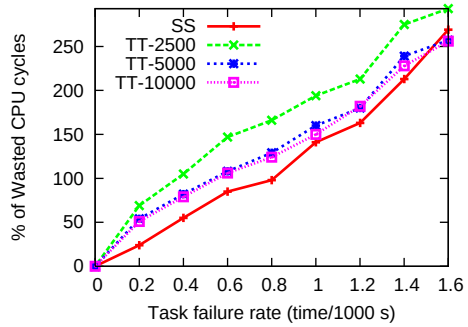

Fig. 6. Plot of the CPU cycles wasted by failed tasks ($\mu = 10000$)

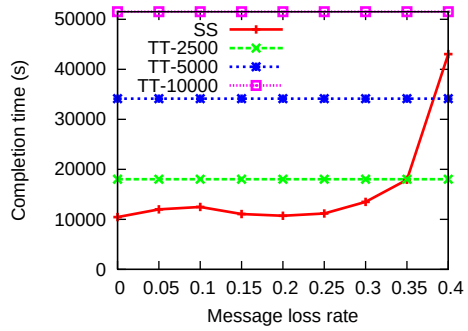

Fig. 7. Plot of the job completion time ($\mu = 10000$)

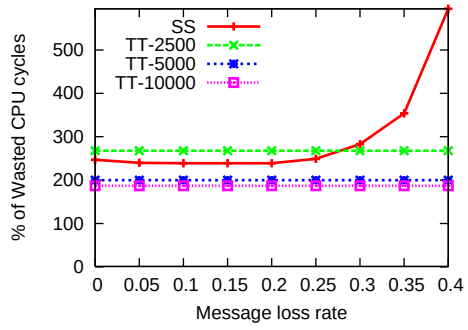

Fig. 8. Plot of the wasted CPU cycles ($\mu = 10000$)

Fig. 2 shows the job completion time when the average task size μ is large. It can be observed that the job completion time of the SS method is short and the impact of the task failure rate is relatively small. In the case of the TT method, the larger the timeout value, the larger the impact of the task failure rate. The reason for this is the difference in the task failure detections in the TT and SS methods. When a task fails, the TT method has to wait for the timeout in order to allocate a new replica to another host. Therefore, the large timeout value delays the start of the replica and also delays the task completion. On the other hand, a scheduler using the SS method can detect task failures immediately because the refresh messages stop when a task fails.

Fig. 3 shows the number of wasted CPU cycles. The impact of task failures in the TT method is almost the same regardless of the timeout value. This is because the replica creation in the TT method is triggered by a timeout and therefore is independent of task failures. On the other hand, the number of wasted CPU cycles in the SS method is almost proportional to the task failure rate. This implies that the number of replicas in the SS method increases with the number of task failures. However, these replicas reduce the job completion time of the SS method and therefore they are accepted as a necessary cost.

Fig. 4 and Fig. 5 show the performances of the two methods when μ is small. The impact of task failures is very small in both the TT and SS methods. The reason for this is that small tasks are not likely to be involved in host crashes because of their short stay in the hosts and they do not waste many CPU cycles even if they fail.

2) Effect of the task timeout value: It should be noted that the performance of the TT method is significantly affected by the task timeout value. Since the timeout value defines the maximum task execution time, the appropriate timeout value depends on the average task execution time. Calculation of the average task execution time requires information such as the average task size and the processing power of hosts. The fact that the performance of the TT method depends on this information contradicts the advantage of WQR, which does not require any information about tasks and hosts.

Moreover, even if we can calculate the average task execution time (denoted as e), the appropriate timeout value for the TT method cannot be determined by a simple calculation such as $e \cdot F$, where F is some fixed value. There are two ways in which the task timeout value affects task execution.

1) If the timeout value is extremely large, the creation of a new replica of a failed task is delayed, and as a result, the completion of the task is also delayed.
2) If the timeout value is very small, many redundant replicas are created and they consume computing resources.

Fig. 2 (when e is large) shows that a timeout value smaller than e is desirable. However, Fig. 4 (when e is small) shows the opposite behavior. These results indicate that effect (1) is dominant for large e and effect (2) is dominant for small e. It is very difficult to estimate the effect of the timeout value on task execution for a given value of e.

3) Effect of task cancellation: It should be noted that in Fig. 3, the number of wasted CPU cycles in TT increases as the task failure rate decreases. The scheduler using TT is not

able to cancel the execution of task replicas even if they are unnecessary. Therefore, even for a low task failure rate, many redundant replicas are executed to the end, and they waste many CPU cycles. On the other hand, the number of wasted CPU cycles in the SS method decreases as the task failure rate decreases. The scheduler using the SS method cancels task replicas as soon as they became unnecessary. Therefore, redundant replicas are cancelled before they waste CPU cycles. This result shows that SS's explicit cancellation of redundant replicas is effective in reducing the wastage of CPU cycles.

When the task failure rate is high, however, the difference between SS and TT is small. This is because a high task failure rate implies that many replicas including redundant ones fail frequently. Fig. 6 shows the wasted CPU cycles of failed tasks. From the graph, it can be observed that most of the wasted CPU cycles are those of failed tasks. In this case, most redundant replicas in SS failed before they were cancelled explicitly.

4) Effect of message losses: We also investigated the effect of changing the message loss rate of the simulations. The average task size is 10000 and the task failure rate is fixed to 1 (time/1000 s).

Fig. 7 and Fig. 8 show the job completion time and the number of wasted CPU cycles, respectively. It is obvious that the performance of the TT method is not affected by the message loss rate because the TT method does not exchange any messages during task execution. The performance of SS is also not affected as long as the message loss rate is lower than a threshold value (approximately 0.3 in the graph). However, as the message loss rate increases above the threshold, the performance of the SS method degrades significantly.

In the SS method, consecutive message losses cause false task abortion (as mentioned in section V-B). If the host aborts its task replica, the scheduler regards the task as failed and allocates a new task replica. However, when the message loss rate is very high, this new task replica may be aborted again. If the message loss rate is higher than the threshold, such abortions occur repeatedly, and the task completion is significantly delayed. The threshold is determined by the refresh interval and the timeout value of the SS method.

It should be noted that this result does not necessarily mean that the SS method is not robust against message losses. A significant degradation in performance occurs only when the high message loss rate continues permanently, which is impractical in real networks. A temporary increase in message losses does not affect the performance of SS.

VII. Conclusions

In this paper, we investigated task scheduling in distributed computing. We selected WQR as the scheduling method because it does not require any kind of information about the hosts or tasks. We used the conventional task timeout method and the soft state method with WQR and compared their performances through simulations. The simulation results are summarized as follows.

- The soft state method is more robust than the task timeout method against task failures.
- The performance of the task timeout method depends on the timeout value, and it is difficult to calculate the appropriate timeout value.
- The soft state method wastes less CPU cycles than the task timeout method when the task failure rate is low.
- There is a threshold for the message loss rate, over which the performance of the soft state method degrades significantly.

From these results, we can conclude that the soft state method is preferable to the task timeout method for task scheduling in distributed computing.

References

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: An Experiment in Public-Resource Computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002.

[2] distributed.net, "Node Zero," http://www.distributed.net/.

[3] F. Dong and S. G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems," No.2006-504, Queen's University School of Computing, Tech. Rep., Jan. 2006.

[4] D. Paranhos, W. Cirne, and F. V. Brasileiro, "Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids," in *Proc. of the EuroPar 2003: Intl. Conf. on Parallel and Distributed Computing*, Klagenfurt, Austria, 2003, pp. 169–180.

[5] D. P. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *Proc. of 5th IEEE/ACM Intl. Workshop on Grid Computing*, Pittsburgh, PA, USA, 2004, pp. 4–10.

[6] W. Cirne, D. Paranhos, F. Brasileiro, and L. F. W. Goes, "On the Efficacy, Efficiency and Emergent Behavior of Task Replication in Large Distributed Systems," *Parallel Computing*, vol. 33, no. 3, pp. 213–234, 2007.

[7] D. P. Anderson, E. Korpela, and R. Walton, "High-Performance Task Distribution for Volunteer Computing," in *Proc. of First IEEE Intl. Conf. on e-Science and Grid Technologies*, Melbourne, Australia, 2005, pp. 196–203.

[8] J. C. S. Lui, V. Misra, and D. Rubenstein, "On the Robustness of Soft State Protocols," in *Proc. of 12th IEEE Intl. Conf. on Network Protocols (ICNP'04)*, Berlin, Germany, 2004, pp. 50–60.

[9] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A Comparison of hard-state and soft-state Signaling Protocols," in *Proc. of the 2003 Conf. on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, 2003, pp. 251–262.

[10] D. Menascé, D. Saha, S. da S. Porto, V. Almeida, and S. Tripathi, "Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures," *Journal of Parallel and Distributed Computing*, vol. 28, pp. 1–18, 1995.

[11] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments," in *Proc. of the 9th Heterogeneous Computing Workshop (HCW'00)*, Cancun, Mexico, 2000, pp. 349–363.

[12] S. Raman and S. McCanne, "A Model, Analysis, and Protocol Framework for Soft State-based Communication," in *Proc. of ACM SIGCOMM*, Cambridge, MA, USA, 1999, pp. 15–25.