VALIDATING DESKTOP GRID RESULTS BY COMPARING INTERMEDIATE CHECKPOINTS*

Filipe Araujo
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
filipius@dei.uc.pt

Patricio Domingues
School of Technology and Management, Polytechnic Institute of Leiria, Portugal
patricio@estg.ipleiria.pt

Derrick Kondo
Laboratoire de Recherche en Informatique/INRIA Futurs, France
dkondo@lri.fr

Luis Moura Silva
CISUC, Department of Informatics Engineering, University of Coimbra, Portugal
luis@dei.uc.pt

Abstract We present a scheme based on the comparison of intermediate checkpoints that accelerates the detection of computing errors of bag-of-tasks executed on volunteer desktop grids. Currently, in the state-of-the-art, replicated task execution is used for result validation. Our method also uses replication, but instead of only comparing results at the end of the replicated computations, we validate ongoing executions by comparing checkpoints of their intermediate execution points. This scheme significantly reduces the time to detect a computational error, which we show with both theoretical analysis and simulation results. In particular, we develop a model that gives the benefit of intermediate checkpointing as a function of checkpoint frequency and error rate, and we confirm this model with simulation experiments. We find that with an error rate of 5% and checkpoint frequency of 20 times per task, the gain is as high as 35% compared to the case where error detection is done only at the end of task execution; for higher checkpoint frequencies or high error rates, the benefits are even greater. In addition, when an erroneous computation is detected at an intermediate execution point, we propose the immediate replacement of that computation with a correct replica from another worker. In this way, useful execution and further validation can continue from that point onward instead of being delayed.

Keywords: Desktop grid, error detection, checkpointing, redundancy

*This work was supported by the CoreGRID Network of Excellence, funded by the European Commission under the Sixth Framework Programme. Project no. FP6-004265.

1. Introduction

Desktop grids, which harvest volunteer computing resources, have gained tremendous momentum in recent years attracting hundreds of thousand of volunteers. Currently, more than a dozen large-scale projects exist, and new ones are being created regularly [6]. The advent of open source and easy-to-setup middleware frameworks like BOINC [4] and XtremWeb [10] have lowered the requirements and skills needed to exploit volunteered resources. To encourage volunteers, projects publish online rankings of contributed work. Interestingly, these rankings cause fierce competition, and attract even more dedicated volunteers [11].

Although desktop grids have a high return-on-investment, they also have major limitations, namely resource volatility and result correctness. The volatility of desktop grids is caused not only by hardware and software faults of computing systems, but also by resource owners who retain full priority in accessing and managing their desktop. Thus, owners reclaiming their resources might force hosted applications to be interrupted. Checkpointing is a common solution to cope with volatility, and some support exists for application-level checkpointing in desktop grid middleware, such as BOINC and XtremWeb [15].

Result correctness of computations performed on volunteer resources is an important issue, since interpreting incorrect results as correct can be worse than no results at all. A major source of result incorrectness is faulty hardware. In [4], Anderson cites *overclocking* as a significant cause of faulty computations in projects that resort to the BOINC framework. The fierce competition and rivalry among volunteers sometimes may also cause unhealthy behavior. Some users try to increase, not always by honest means, their credits. In some extreme cases, users resort to dishonest tricks to collect undue credits, like fabricating results that require much less computation than the real ones [12]. These users are known as *lazy cheaters*. Finally, another type of malicious user – saboteur – might simply act for the sole purpose of ruining the computation, without concern for credits [14]. In contrast to lazy cheaters, saboteurs may be difficult to counter since they may be resourceful and committed to perform everything they can to disrupt the computation.

Commonly, desktop grid projects resort to redundancy as a sabotage-tolerance technique [8]. Under this approach, the same task is distributed to r different worker machines (hopefully unrelated) to avoid collusion. When completed, results are compared and there is a majority vote. If a result has majority, that is, more than $r/2$ tasks return this result or an equivalent one [1], it is interpreted as the correct one and the task is flagged as completed. On the

[1] Some projects dependent on floating-point operations might have slightly different results when executed in different platforms, but yet equivalent from the project point-of-view [17].

contrary, if no consensus can be found, all results are discarded and the task is marked for rescheduling.

In this paper, we present a checkpoint and replication-based error detection technique that exploits checkpointing and redundancy. The technique compares intermediate checkpoint digests of redundant instances of a same task. If differences are found, the conclusion is that at least one execution is wrong. In contrast to the simple redundancy mechanism, where diverging computations can only be detected after a majority of tasks have completed, intermediate checkpoint comparison allows for earlier and more precise detection of errors, since execution divergence can be spotted at the next checkpoint following any error. This allows one to take proactive and corrective measures without having to wait for the completion of the tasks, therefore permitting faster task completion, since faulty tasks can immediately be rescheduled.

To complement the checkpoint-based comparison methodology for error detection, we propose a checkpoint-based replication technique whose goal is to promote fast completion of redundant instances of a same task, in order to speed up validation of results. Specifically, under the proposed technique, the replication of a redundant instance is scheduled as soon as the instance is determined to be erroneous or lagging behind. To minimize the computation to be redone, the technique tries to initialize the replica from a validated intermediate checkpoint. The technique extends the checkpoint-based verification, promoting a balanced execution of redundant instances, since validation can only occur when a majority of results have been completed. Moreover, since credits are given to workers only after results have been validated, this also accelerates validation and proper credit assignment, which is an important issue for a considerable percentage of volunteers [11].

Specifically, the contributions of the paper are as follows. First, we construct a model that estimates the benefit of comparing intermediate checkpoints as a function of the probability of task error and checkpoint frequency. Second, we propose the use of immediate replacement of erroneous or slowly executing tasks to prevent delays of task execution and validation. Third, we conduct simulations and analysis of results using our novel approach, which confirms the benefits estimated by our theoretical model.

The remainder of this paper is organized as follows. In Section 2, we define the assumptions used by the comparison techniques that are based on checkpointing and replication. In Section 3, we present our technique for error detection through checkpoint comparison and our theoretical model, while in Section 4, we introduce checkpoint-based task replication. In Section 5, we describe our simulation setup and results. In Section 6, we discuss related work. Finally, in Section 7, we summarize the conclusions and describe future work.

2. Assumptions and Definitions

We assume a large-scale computing project, where a central supervisor coordinates the whole computation, by distributing tasks to requesting volunteer worker machines (henceforth *workers*). The tasks that comprise an application are sequential and independent from each others. Furthermore, we assume that all communications occur exclusively between workers and the supervisor. To circumvent Internet asymmetries [16] caused by NAT and firewall schemes, communications are worker-initiated. Thus, the supervisor is passive in the sense that it can only answer to worker requests. Note that this communication model is the one adopted by several desktop grid frameworks [10, 4].

At the worker level, fault-tolerance is achieved through application-level checkpointing [15]. We only consider tasks which can individually be broken into m temporal segments $S_t = \{S_{t_1}, \ldots, S_{t_m}\}$. The intermediate computational states can be checkpointed at the end of each temporal segment, yielding the checkpoint set $C = \{C_1, \ldots, C_m\}$, with C_m taken at the end of the computation. Projects with long duration tasks (weeks or months long), such as for example the climateprediction.net, whose tasks last for months on state-of-the-art machines, can benefit most from checkpointing. Whenever a task is interrupted (because the user switches the machine off, or for some other reason), its execution can be resumed from the last stable checkpoint C_j.

Depending on the application, checkpoints can get quite large, in the range of tens to hundreds of megabytes in size, and thus it might be inefficient to transfer and compare them. (For the purpose of comparison, all checkpoints need to be on the machine that effectively performs the comparison; thus at least one of them has to be transferred.) For comparison purposes, we assume that message digests of checkpoints (provided by the MD5 [13] and the SHA-family [9] algorithms, for example) can be used. Due to their reduced and predictable dimensions, message digests can be easily exchanged and compared. Furthermore, an application-specific pre-processing function might be deployed to normalize checkpoints (for instance, for removing task-dependent identifiers) prior to the use of a generic digest algorithm. For the purpose of comparison, checkpoint C_j is represented by the message digest $\mathrm{MD}(C_j)$. Additionally, the comparison of checkpoints needs to be executed between what we term as *equivalent checkpoints*, that is, checkpoints from different replicas of a task that represent a same execution point of the task.

Regarding redundancy, we assume that the system executes each task r times, by r independent workers, with the supervisor applying majority voting to validate results, electing the so-called *canonical result* [4]. Afterwards, when the result verification is completed, the system assigns the proper credits to the workers which have returned correct results.

3. Comparison of Equivalent Checkpoint Digests

For the comparison of equivalent checkpoint digests, a worker is requested to return, along with the results of the task that it computed, a selected set of message digests of the checkpoints saved during the task computation. The list of checkpoints whose message digests are requested is defined at task creation time so that redundant instances of a task share the same set of requested checkpoint digests.

When a majority of redundant executions are completed, and the supervisor holds enough results for meaningful comparisons, the checkpoint digests from equivalent execution points are compared to each others. If the digests are different, the execution point where the differences were detected is marked as suspicious. Comparatively to the sole result comparisons, the selective digests technique permits a finer grain detection level, since an erroneous computation can be located right after the first divergent checkpoint.

3.1 Reducing the Time to Detect an Error

Although the selective digests strategy allows for a more precise location of error occurrence, it does not speed up the detection of incorrect computations, since error detection can only occur after, at least, two redundant instances have terminated.

A more proactive variant is to have workers returning available checkpoint digests during the computation. Ideally, from a detection point-of-view, the worker should send to the supervisor a checkpoint digest immediately after its computation. This way, an error can be spotted by the supervisor as soon as a majority of checkpoint digests is available for the considered execution point. Thus, upon detection of a divergent computation, corrective measures can immediately be triggered by the supervisor. For instance, an additional instance of the task can be scheduled. Additionally, the thought-to-be faulty worker can be marked as a suspect and further probed to assess its computational honesty, or, if repeating a faulty behavior it can be blacklisted altogether [14].

3.2 Theoretical Analysis

In this section, we conduct an initial analysis of the advantage of detecting erroneous computations at intermediate checkpoints. The goal of this analysis is to estimate the potential advantages of our approach.

We assume that a task is segmented into m fragments. Additionally, we make the following simplifying assumptions: (1) machines and segments are homogeneous: a segment always takes t time to complete and the entire task requires $T = m \times t$. Hence, the number of segments, m, determines the computational effort of the task. The probability of obtaining a wrong checkpoint

is the same for all the workers and for all checkpoints of the same task; (2) all the replicas of a task start at the same time across all workers; (3) the errors are independent of each others, and thus, no contamination of replicas occur, meaning that comparison of replicas is enough to catch all the errors.

Although these assumptions may seem too restrictive, we show experimentally in Section 5 that our analysis also holds for other more heterogeneous scenarios. We will focus on two variables that affect the system: the probability, p_e, of having a computational error in any of the checkpoints (either due to a computational mishap or malicious behavior) and the number of checkpoints of the task. We consider that results are validated through r-replication. (All the replicas must compute the same equivalent checkpoint digest.) However, comparison of intermediate checkpoint digests permits partial validation at point j as soon as the r replicas of a task have sent back their respective message digests of checkpoint j, that is, MD(C_j). We compare this new and improved approach against the state-of-the-art method, which can only detect an error at the end of the execution.

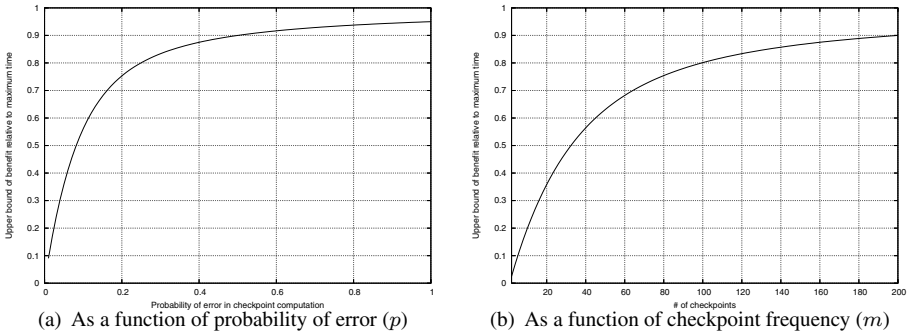
When the computational error occurs before the first validation checkpoint (C_1), the checkpoint comparison method will permit a detection $T \cdot \frac{m-1}{m}$ time units sooner than the regular methodology. This case occurs when there is one or more errors in the computation of all the r replicas. It is easy to see that the probability of this event is $1 - (1 - p_e)^r$, which we denote as p to simplify. For the next checkpoint, the comparison of equivalent checkpoints saves $T \cdot \frac{m-2}{m}$ time units, relative to the normal validation method. This occurs with probability $p \cdot (1 - p)$. Extending this reasoning to checkpoint i yields a saving of $T \cdot \frac{m-i}{m}$ with probability $p \cdot (1 - p)^{i-1}$. (In the last segment, when $i = m$, or if there is no error for the whole computation, our approach brings no benefit.) We let W be a random variable to represent the error detection time, that is, the time elapsed from the occurrence of an error up to its detection. In other words, if we reschedule the task as soon as the error in the checkpoint is detected, W represents the maximum time that we can save, relatively to the compare-at-end approach, with a single error detection. However, in the regular strategy, the computation time can be even worse than $T + W$, because other errors can delay the task even further. Hence, if we are able to calculate W, we can have a measure of the advantage of detecting errors by comparing intermediate checkpoints. To calculate the expected value of W, we proceed as follows (we omit the probability of not having any error, as there is no gain in that case):

$$E[W] = \sum_{i=1}^{m} \left(pq^{i-1} \cdot \frac{m-i}{m} T \right) = Tp \left(\sum_{i=1}^{m} q^{i-1} - \frac{1}{m} \sum_{i=1}^{m} i \cdot q^{i-1} \right) \quad (1)$$

Where $q = 1 - p$. Since $\sum_{i=1}^{m} q^{i-1}$ is a sum of terms of a geometric sequence, its sum is $S_{m-1} = \frac{1-q^m}{1-q}$. We can use standard techniques to compute the second term of the difference. Consider that $S'_{m-1} = \sum_{i=1}^{m} i \cdot q^{i-1}$. By multiplying S'_{m-1} by q and taking the difference $(1-q)S'_{m-1}$, we get $S'_{m-1} = \frac{S_{m-1} - mq^m}{1-q}$. Since $p = 1 - q$, this yields:

$$E[W] = TpS_{m-1} - \frac{Tp}{m} S'_{m-1} = T\left(1 - \frac{1-q^m}{mp}\right) \tag{2}$$

In Figures 1(a) and 1(b) we depict the time that we can save relative to T ($E[W]/T$), considering Equation 2. In Figure 1(a), we set $m = 20$, while in the other figure we set $p = 0.05$. From Eq. 2 we conclude that the maximum time that a checkpoint comparison can save converges to T, when $m \to \top$. When $p \to 1$, the time that we can save approaches $T \cdot \frac{m-1}{m}$ as we would expect. For example, we find that with only an error rate of 5% and checkpoint frequency of 20 times per task, the gain is as high as 35% compared to the case where error detection is done only at the end of task execution. Note that this is a conservative estimate of the benefit as many projects (such as Einstein@home and SIMAP [2–3]) checkpoint more often in a given work unit. In particular, in the BOINC project climateprediction.net [7], a work unit requires around 3 months of CPU time in a fast PCs, being checkpointed 72 times during the whole execution. In conclusion, for even conservative estimates of error rates and checkpoint frequencies, the benefit of comparing digests of intermediate checkpoints is significant, and is even greater for higher probabilities of error or for longer computations with checkpoints.



(a) As a function of probability of error (p) (b) As a function of checkpoint frequency (m)

Figure 1. Benefit (W) relative to maximum time (T).

4. Checkpoint-based Task Replication

Some (BOINC-based) desktop projects increase, at least for specific period of times, the redundancy level to foster the chance of fast completion of tasks. Surprisingly, one of the main motivation for this important decision is not directly related to the gain of an higher confidence level for the results, but the need to quickly rewards worker with the proper amount of credits. In fact, credits are only committed to workers after validation of the sent results. These credits are determined by the supervisor, based on the credit claims made by the intervening workers (jointly with the completed results, the worker sends a claim with the amount of credits it believes it deserves). To circumvent the high volatility of volunteers, a number of instances higher than what is required for majority voting is scheduled for execution. This provides timely assignment of credits even in the presence of sluggish and drop-out workers. However, this approach wastes resources, and slows down the whole computation.

To speed up completion and validation of individual tasks, promoting fast credit assignment, we propose to combine the comparison of intermediate checkpoint digests with task replication. To prevent lengthy re-computations due to the replication of task, we resort to validated checkpoints to load execution state in tasks to replicate, avoiding to restart from scratch.

The task replication works by loosely coupling the execution of the redundant instances of a same task, which are configured for reporting selective checkpoint digests. Note that workers processing instances are not aware of each other (otherwise the risk of collusion would increase). The supervisor follows the progress of the coupled instances of a task through the messages holding the checkpoint digests sent back by these instances, validating the received checkpoint digests of the selected execution point through comparison as soon as a majority of results has been received.

Whenever a worker lags behind its instance partners by more than a specified threshold – the threshold takes into account the relative speed of the workers – the supervisor initiates a replace operation, with the goal of substituting the behind-schedule worker. To further speed up substitution, the substitute task should start from the last validated checkpoint, if available. To prepare for the instance substitution, the supervisor requests, upon the next communication of a paired-worker, the last validated checkpoint from this worker (not the digest, the entire checkpoint file). Upon receiving it, it checks its validity through message digest comparison, and creates a task which integrates the validated checkpoint file. This *replace task* is then scheduled to a requesting worker, which starts the computation from the checkpoint execution point, thus skipping the computation up to this point. From the point of view of the supervisor, the newly scheduled task replaces the lost/delayed one, and thus the monitoring of execution proceeds as previously explained. Note that, in

order to prevent excessive replicas, replication should only be performed if the number of instances is below a predefined threshold.

5. Experimental Results

In this section, we confirm and extend the theoretical results obtained in Section 3.2 through simulation. Specifically, we assign a number of tasks to a set of workers, setting the duration of these simulated tasks beforehand. Whenever a worker computes a checkpoint, it randomly determines whether that computation is wrong or correct (once a checkpoint is wrong, all the remaining checkpoints from that worker are also considered as wrong.) The total time of the computation, T, is the time at which the last replica finishes its last checkpoint, regardless of whether it is correct or wrong. Assume that checkpoint C_j was the first one to be wrong and that the last replica finished C_j at time T_W. We are interested in the random variable $W = T - T_W$, which represents the benefit of using intermediate checkpoints relative to the state-of-the-art. In particular, the metric we use to quantify the gain compared to the state-of-the-art is the relative value W/T.

We started by considering the same parameter settings that were used to generate Figure 1(b). So, we set an uniform $p_e \approx 0.0253$ for all execution segments, considering homogeneous segments, and a two-replica scheme, which corresponds to $p = 0.05$. As expected, we got a curve that closely follows the theoretical prediction. Then, we studied the impact of considering different durations for the checkpoints and different error probabilities for each of the computed checkpoints. We used two different random distributions for this: uniform and truncated Gaussian. To maintain consistency, the average values for the error probability and for the segment duration were the same as for the fixed case, p_e and T, respectively. In the uniform distribution, the actual error probability was chosen uniformly from the interval $[0.5p_e, 1.5p_e)$ (which is always inside the interval $[0, 1]$), while the duration was chosen using the same distribution in the interval $[0.5T, 1.5T)$. For the Gaussian distribution, we considered averages of p_e and T, and standard deviations of 30% of the average. Additionally, we truncated the values of p_e and T to be inside the ranges $[0.5p_e, 1.5p_e]$ and $[0.5T, 1.5T]$, respectively. In Figure 2, we show the average result of varying the number of checkpoints for 300 different trials. As we can see, the curves overlap.

The most interesting conclusion from these results is that the particular random distribution that controls the duration and the errors of the checkpoints does not seem to make any significant difference, at least for the same averages. This would not be true if, for instance, the average duration of checkpoints

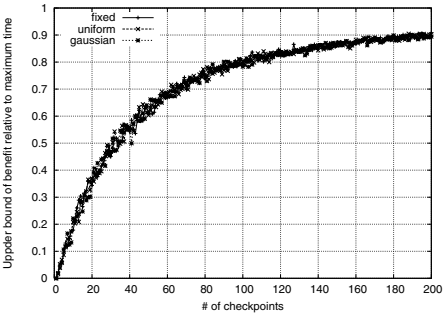


Figure 2. Benefit (W) relative to expected maximum time (T) (obtained experimentally).

i and j was different for checkpoints i and j[2]. We believe that there is a simple and intuitive reason for this; on average the slowest replica should finish checkpoint i around time $i \cdot \frac{T}{m}$, where T is the time at which the slowest replica finishes the task. Although some particular cases may not follow this trend, our experimental results confirm this intuition for the average case.

6. Related Work

Antonelli et al. [5] propose a distributed checkpoint-based technique for sabotage tolerance addressing sequential computation split in multiple consecutive temporal segments. To certify a given checkpoint C_j, the supervisor creates a verification task that references the checkpoint to verify and holds the network contact details of the worker which performed the computation. The task is then assigned to a worker node (*verifier*), which requests the checkpoint from the worker being scrutinized, and loads it upon reception, executing the task up to the next checkpoint, that is, C_{j+1}. It then sends the message digest of the this checkpoint to the supervisor. Finally, the supervisor compares the digest to the other equivalent digests. The scheme is appealing since it distributes the computation needed for verification of checkpoints through the workers. However, some major issues like asymmetrical communications and node availability are not addressed by the authors. Furthermore, workers need to keep some of the checkpoints of the computed tasks and transfer them when requested, a demand that might require meaningful space storage and network bandwidth, especially with large individual checkpoints. On top of that, promoting direct contact between workers may ease collusion.

[2]However, note that it would not make much sense to consider different average durations for different checkpoints, unless we were targeting a particular application with a well-known behavior.

Agbaria and Friedman [1] propose a replication and checkpoint-based scheme to detect intrusions through anomaly spotting. They resort to checkpoint comparison for the purpose of identifying intrusions in a Byzantine environment. Similarly to our approach, the execution is split in n sequential phases, with a checkpoint being taken by each worker node at the end of each phase. For supporting a maximum of t intruded nodes (each node executes a replica), the proposed scheme requires $t + 1$ replicas when no intruded node exists. However, when intrusion exists, the protocol needs additional stages, involving more than the $3t + 1$ replicas which would be required by a straight Byzantine agreement protocol. The unbalance is supported by the fact that intrusions are rare and thus it compensates to have a lightweight scheme which is only penalized when intrusions do occur. The protocol distinguishes between *workers* (nodes that perform the computation and which can get intruded) and *auditors*, which are responsible for assessing the integrity of the workers. Specifically, the auditors are used to agree that all the $t + 1$ replicas match. A major requirement of the protocol lies in the required synchronization, with workers having to send their checkpoints to the auditors within a given time frame. This requires that the replica execution occurs simultaneously, a premise that might hard to fulfill in a volatile environment such as desktop grids. Furthermore, the checkpoints (or equivalently, a message digest) need to be sent to the auditors at the end of every stage, an operation that requires communication resources and might be difficult if auditors are not directly addressable [16]. Relatively to the solution that we propose, our emphasis is more on the practicality of the error detections schemes and its integration with current desktop grid frameworks.

7. Conclusion

We proposed a strategy for early detection of errors by comparing checkpoints of redundant tasks executed over desktop grid resources. We developed a theoretical model that estimates the benefit of using intermediate checkpoints given a task length and task segment error rate. We confirmed this theoretical analysis with simulation results. We find that with only an error rate of 5% and checkpoint frequency of 20 times per task, the gain is as high as 35% compared to the case where error detection is done only at the end of task execution. For higher checkpoint frequencies or high error rates, the benefits are even greater.

For future work, we plan to extend the study the case where segments are completed with non-uniform execution times. In addition, we will study and characterize work unit error rates in a real BOINC project, namely XtremLab [18], and then instantiate our model with such error rates. Finally, we intend to study the use of trickle messages [7] to regularly send the checkpoint digests to the central supervisor, without incurring any additional communication costs.

References

[1] A. Agbaria and R. Friedman. A replication-and checkpoint-based approach for anomaly-based intrusion detection and recovery. *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 137–143, 2005.

[2] D. Allen. Personal communication, June 2006.

[3] C. An. Personal communication, March 2006.

[4] D. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004.

[5] D. Antonelli, A. Cordero, and A. Mettler. Securing Distributed Computation with Untrusted Participants. 2004.

[6] J. Bohannon. Grassroots supercomputing. *Science*, 308(6 May):810–813, 2005.

[7] C. Christensen, T. Aina, and D. Stainforth. The challenge of volunteer computing with lengthy climate model simulations. In *1st IEEE International Conference on e-Science and Grid Computing*, pages 8–15, Melbourne, Australia, 2005. IEEE Computer Society.

[8] W. Du, J. Jia, M. Mangal, and M. Murugesan. Uncheatable grid computing. *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 4–11, 2004.

[9] D. Eastlake and P. Jones. RFC 3174: US Secure Hash Algorithm 1 (SHA1). *Request for Comments, September*, 2001.

[10] G. Fedak, C. Germain, V. Neri, and F. Cappello. Xtremweb: A generic global computing system. In *1st Int'l Symposium on Cluster Computing and the Grid (CCGRID'01)*, pages 582–587, Brisbane, 2001.

[11] A. Holohan and A. Garg. Collaboration Online: The Example of Distributed Computing. *Journal of Computer-Mediated Communication*, 10(4), 2005.

[12] D. Molnar. The SETI@home Problem. *ACM Crossroads Student Magazine*, september 2000.

[13] R. Rivest. RFC-1321 The MD5 Message-Digest Algorithm. *Network Working Group, IETF*, April 1992.

[14] L. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. In *1st International Symposium on Cluster Computing and the Grid*, page 337, 2001.

[15] L. M. Silva and J. G. Silva. System-level versus user-defined checkpointing. In *Symposium on Reliable Distributed Systems*, pages 68–74, 1998.

[16] S. Son and M. Livny. Recovering Internet Symmetry in Distributed Computing. *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 542–549, 2003.

[17] M. Taufer, P. J. Teller, D. P. Anderson, and I. Charles L. Brooks. Metrics for effective resource management in global computing environments. *e-science*, 0:204–211, 2005.

[18] XtremLab. `http://xtremlab.lri.fr`.

http://www.springer.com/978-0-387-72811-7

Achievements in European Research on Grid Systems
CoreGRID Integration Workshop 2006 (Selected Papers)
Gorlatch, S.; Bubak, M.; Priol, T. (Eds.)
2008, XVIII, 238 p. 20 illus., Hardcover
ISBN: 978-0-387-72811-7