Elastic Management of Byzantine Faults

Luciana Arantes, Marjorie Bournat, Roy Friedman, Olivier Marin, Pierre Sens

Elastic Management of Byzantine Faults

Luciana Arantes*, Marjorie Bournat*, Roy Friedman †, Olivier Marin *, and Pierre Sens *
*Sorbonne Universités, UPMC Univ Paris 06,
CNRS, Inria, LIP6
F-75005, Paris, France
Email: firstname.lastname@lip6.fr
†Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
Email: roy@cs.technion.ac.il

***Abstract*—Tolerating byzantine faults on a large scale is a challenge: in particular, Desktop Grid environments sustain large numbers of faults that range from crashes to byzantine faults. Solutions in the literature that address byzantine failures are costly and none of them scales to really large numbers of nodes. This paper proposes to distribute task scheduling on trusted nodes in a Cloud network and to have these nodes assess the reliability of worker nodes by means of a reputation system. The resulting architecture is built for scalability and adapts costs to the workload associated with client requests.**

I. Introduction

Desktop Grids such as BOINC (Berkley Open Infrastructure for Network Computing) [1] follow a master/worker approach to provide access to an important number of commodity nodes, hence they are well suited for very large computations which can be divided into small tasks. However, commodity nodes are not reliable: they are very likely to incur faults that range from crashes to byzantine faults. Workers that sustain byzantine faults may return incorrect results and therefore corrupt the whole computation.

Replication provides an efficient workaround to byzantine faults: several workers compute the same task concurrently, and if enough workers return the same output it is considered correct. The state of the art distinguishes two categories of solutions based on replication: deterministic [2], [3], [4], [5] and probabilistic methods [6], [7]. Deterministic solutions make the very strong assumption that there are at most f byzantine faults in the system, and replicate every task on a fixed number of workers. But choosing the right value for f is highly complex. If f is much larger than the real number of byzantine faults in the system, then the solution is far from cost efficient. Conversely, setting f too small increases the likelihood of the solution to return incorrect results. Probabilistic solutions don't make any such assumption: instead, they adapt the degree of replication with a heuristic formula that estimates the likelihood of a correct result.

Regardless of whether they apply deterministic or probabilistic replication, byzantine-tolerant solutions generally do not scale because they rely on a unique and trusted scheduler. In the case of BOINC, a static cluster of trusted schedulers with a consensus protocol handles client queries, which improves scalability. However, its scalability remains limited because of the fixed number of schedulers and it uses a deterministic replication which is not cost efficient.

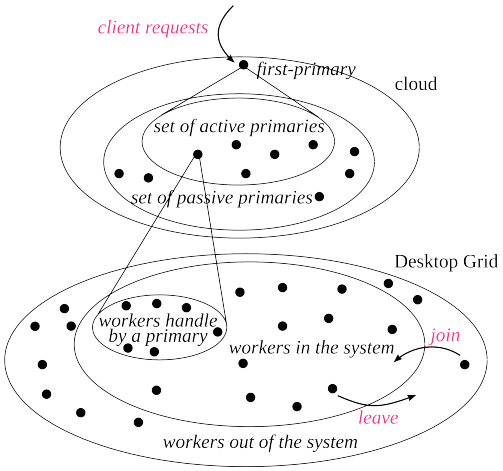

Fig. 1. Architectural design of our elastic Desktop Grid

This position paper presents our proposed approach: an elastic extension of the BOINC desktop grid model on top of a cloud service where trusted schedulers use a reputation system to implement a probabilistic replication scheme. The principle of elasticity is to adapt resource provisioning as a means to optimize the tradeoff between cost and performance. Our system scales in and out as it adapts the number of provisioned schedulers to the number of workers. It also improves cost efficiency by maintaining worker reputations and enforcing probabilistic replication in order to minimize the number of workers allocated to each task.

II. Design for an Elastic Desktop Grid

Figure 1 presents our design for a distributed architecture where the number of trusted schedulers provisioned in the cloud, called *primaries*, evolves with the number of available commodity nodes that act as workers. Each *primary* handles a subset of workers, and the intersection between subsets handled by two different *primaries* remains empty. We introduce a reliable node, called *first-primary*, which maintains the list of all active *primaries*. The role of the *first-primary* is limited to dispatching client requests to *primaries*: a centralized entity should not hinder scalability. If the first-primary becomes a bottleneck, the dispatch can be handled equally well by a larger number of *first-primaries*.

Our elastic architecture uses probabilistic replication: it may return incorrect results. Nevertheless it is possible to assess the reliability of a result: each primary maintains a distinct reputation for every worker it handles. A reputation is a subjective estimation of the correctness of previous answers returned by a worker, and provides some insight about the correctness of future answers. A scheduler assigns a default reputation value to every new worker that joins its handled subset. This reputation will evolve: if the worker returns the same answer as a majority of workers that compute the same task, then its reputation increases. Conversely its reputation decreases if its result diverges from the majority answer. A heuristic formula allows every scheduler to calculate the reliability of a task result as a function of the number of workers that compute the task and of their reputations.

This architectural design enables the application of a wide range of strategies. To start with, there are different ways to update a reputation. For example the reputation value of a worker can equal the rate of its answers deemed correct. Another possibility is to increase and decrease the reputation value with fixed reward and punishment ratios. There are also many ways to form replication groups assigned to a task, the goal being to form a group with a collective reputation high enough to deliver a correct answer with a high probability. For instance the scheduler can just add workers randomly until it achieves its target reliability for the group, or it can sort workers according to their reputations and form smaller groups with the most reputable workers. When a group of workers fails to achieve a majority answer, the scheduler must assign additional workers to this task. This can be done by considering the reputation of workers or by considering the gap between the number of correct and wrong answers. We have compiled a collection of such strategies from two previous works: Sonnek et al. [6] and Arantes et al. [7]. We also intend to apply two incremental replication schemes: iterative and progressive redundancy [8].

The distribution of workers into scheduler subsets offers yet another range of strategies. Subsets can be arranged to hold workers with specific ranges of reputation values, or worker reputations can be distributed uniformly among the subsets. In the former strategy, clients can request a primary (via the first-primary) in a random way or considering a reputation range.

We intend to analyze and compare the performance of our architecture when applying these different strategies.

III. Assessment of our architecture

We identify 5 main metrics for assessing our solution: *response time*, *throughput*, *reliability*, *scalability*, and *cost efficiency*.

We aim for a solution that processes as many client requests as possible in the shortest possible amount of time. We associate two metrics with this aim: the average *response time*, that is the time elapsed between the submission of a client request and the reception of the output, and the *throughput*, the number of requests that get processed within a given time interval. We also want a solution that tolerates byzantine faults. We will evaluate its *reliability* by calculating the percentage of correct answers it outputs. To assess the *scalability* of our solution, we will gradually increase the number of workers and see how it impacts *response time* and *throughput*. Finally we will assess *cost efficiency* with the average degree of replication for every task weighted by the reputation of the allocated workers.

IV. Conclusion

This paper studies the scalable application of replication techniques to tolerate byzantine faults on a Desktop Grid. We introduce a new elastic architecture which scales with the number of commodity nodes and uses probabilistic replication to improve cost efficiency. We also define different strategies that can be applied to our architecture, along with metrics to assess these strategies.

We have recently finalized a running implementation of our architecture on top of the SimGrid simulator [9]. Our implementation integrates several of the proposed strategies mentioned in this paper. Our next step is to conduct simulation experiments in order to analyze and compare the performance of our architecture when applying these different strategies.

Acknowledgments

This work was supported by a grant from CAMPUS France and the Israeli Ministry of Science and Technology PHC-Maimonide, grant #31807PC.

References

[1] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *Proc. of the 5th IEEE/ACM International Workshop on Grid Computing*, Washington, DC, USA, 2004, pp. 4–10.

[2] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), USA, 1999*, 1999, pp. 173–186.

[3] R. Guerraoui, N. Knezevic, V. Quéma, and M. Vukolic, "The next 700 BFT protocols," in *Proceedings of the 5th European conference on Computer systems, EuroSys 2010, France*, 2010, pp. 363–376.

[4] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. L. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, USA, October 14-17, 2007*, 2007, pp. 45–58.

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

[6] J. D. Sonnek, A. Chandra, and J. B. Weissman, "Adaptive reputation-based scheduling on unreliable distributed infrastructures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 11, pp. 1551–1564, 2007.

[7] L. Arantes, R. Friedman, O. Marin, and P. Sens, "Probabilistic Byzantine Tolerance for Cloud Computing," in *34th International Symposium on Reliable Distributed Systems (SRDS'15)*, 2015.

[8] Y. Brun, G. Edwards, J. Y. Bang, and N. Medvidovic, "Smart redundancy for distributed computation," in *2011 International Conference on Distributed Computing Systems, ICDCS 2011, USA*, 2011, pp. 665–676.

[9] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014.