Toward Real-time, Many-Task Applications on Large Distributed Systems

Sangho Yi[1], Derrick Kondo[1], and David P. Anderson[2]

[1] INRIA, France
{sangho.yi,derrick.kondo}@inrialpes.fr
[2] UC Berkeley, USA
davea@ssl.berkeley.edu

Abstract. In the age of Grid, Cloud, volunteer computing, massively parallel applications are deployed over tens or hundreds of thousands of resources over short periods of times to complete immense computations. In this work, we consider the problem of deploying such applications with stringent real-time requirements. One major challenge is the server-side management of these tasks, which often number in tens or hundreds of thousands on a centralized server. In this work, we design and implement a real-time task management system for many-task computing, called RT-BOINC. The system gives low $O(1)$ worst-case execution time for task management operations, such as task scheduling, state transitioning, and validation. We implement this system on top of BOINC, a common middleware for volunteer computing. Using micro and macro-benchmarks executed in emulation experiments, we show that RT-BOINC provides significantly lower worst-case execution time, and lessens the gap between the average and the worst-case performance compared with the original BOINC implementation.

1 Introduction

Workloads on Grid, Cloud, and Volunteer Computing platforms often consist of tens or hundreds of thousands of parallel tasks that must be processed in short amounts of time on the order of hours or days [1]. In this work, we focus on real-time applications of similar size where the deadline per task is on the order of seconds or tens of seconds. Important applications include online strategy games (such as Go [2] or Chess [3]), interactive visualization [4] (possibly with precedence constraints), and real-time digital forensics [5].

Our aim to enable the execution of real-time applications on large (on the order of 10,000 nodes) distributed systems, such as volunteer computing platforms. Volunteer computing platforms use the free resources in Intranet and Internet environments for large distributed computation, and currently provide over 8 PetaFLOPS of computing power for over 50 applications. However, these applications are limited to mainly high-throughput jobs or large batch jobs.

There are three main challenges for supporting real-time applications. First, one must ensure or predict the availability of volunteers. This has been the subject of recent work described in [6–8]. Second, one must bound network latency

between the server and clients. Much work also exists on network distance that one could leverage [9]. Third, management of hundreds of thousands of tasks on the server must be efficient and in particular, have bounded execution time.

In this work, we focus on the third challenge. Server-side management includes task generation, the transition of task and result states, and scheduling. These management functions can incur significant overheads, given that the desired job makespans are on the order of seconds and the number of tasks per job in on the order of tens of thousands.

The main contribution of this work is the design and implementation of a real-time management system, based on a popular middleware for volunteer computing called BOINC. Our system RT-BOINC gives low worst-case bounds on server-side task management, while minimizing the gap between worst-case and average execution time. Our approach is to use novel data structures (in particular multi-level lookup tables) and functions that ensure O(1) worst-case complexity. In emulation experiments with our prototype, we show performance improvements of often 2 orders of magnitude compared to the original BOINC.

The remainder of this paper is organized as follows. Section 2 describes related work in volunteer computing environments. Section 3 presents the design and internal structures, and implementation of RT-BOINC. Section 4 evaluates performance of RT-BOINC and the original BOINC in terms of both the average and worst-case execution time. Finally, Section 5 presents conclusions and future work.

2 Related Works

Volunteer computing systems, such as XtremWeb and Condor, are tailored for maximizing task throughput, not minimizing latency on the order of seconds. For instance, in [1], Silberstein et al. proposed GridBot, which provides efficient execution of *bags-of-tasks* on heterogeneous collections of computing platforms including grid, volunteer, and cluster computing environments virtualized as a single computing host. While the system uses a hierarchical task management system, the system cannot provide the task-level guarantees of execution time.

Hierarchical systems can improve server performance but they still often do not provide any guarantee of performance in terms of worst-case execution time. For instance, in [10], Kacsuk et al. proposed a SZTAKI desktop grid, which is a hierarchical system developed on top of BOINC server structures. They modified the original BOINC server to have multiple levels of workunit distribution. By doing this, SZTAKI can reduce load on the primary BOINC server by using the second and third-level BOINC servers. But, each level of BOINC servers still has the same characteristics of the original BOINC, which performance is not guaranteed.

Dedicated supercomputers can run real-time tasks, but volunteer computing could be an low-cost alternative if it could support real-time guarantees. In the domain of complex strategy games, Deep Blue [3] was the first machine defeat the human world champion in 1996. IBM developed a dedicated server system

for Deep Blue, and the server achieved about 11.38 GFLOPS on the LINPACK benchmark. Since 2006, several researchers in the world have been developed MoGo, which is software to find the next move in the game of Go. They adapted Monte-Carlo-based algorithms, and now, they are as strong as the professional Go players in the 9×9 small board based on the cluster computing machines[2].

Grid gaming middlewares [11] have been developed and address issues such as adaptive redirection of communication or computation given variable load. They also address issues such as high-level easy-to-use programming interfaces, and monitoring tools for capacity planning. We believe our work on giving worst-based bounds on execution time is complementary with those methods. For example, our techniques guarantee performance given that the data can be store entirely in the server's memory; this in turn could be used with capacity planning tools to determine when to replicate a server.

3 Design and Implementation of RT-BOINC

In this section, we briefly describe the internal structures of BOINC, and we present some requirements for computing real-time and interactive tasks in volunteer computing environments. Then, we present the design and implementation of RT-BOINC in detail.

3.1 The original BOINC

BOINC server consists of two main parts, namely the main database, and server daemon processes (*feeder*, *transitioner*, *assimilator*, *validator*, *file-deleter*, *work-generator*, and *scheduler*).

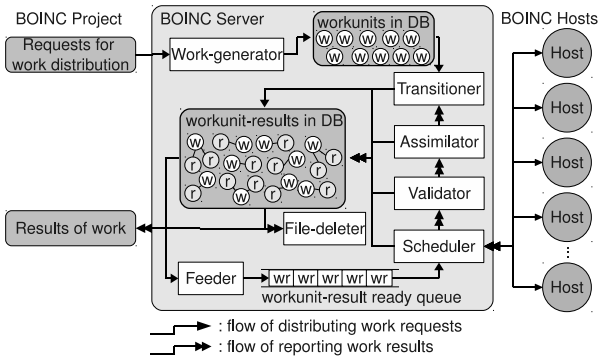

Fig. 1. Internal structures of BOINC server

When the *project-manager* sends work for the BOINC server, the *work-generator* creates several *workunits* on the main database. Then, *transitioner*

makes *workunit-results* pairs on the database. The pairs are fed to the *scheduler* by the *feeder*, and they are distributed to multiple BOINC hosts. When each host returns its result, the *scheduler* reports it to the *validator*. When the validation is completed, the data will be processed and finalized by the *assimilator*. Finally, the *project-manager* can get the assimilated results from BOINC server.

BOINC projects use the same structures described in Fig. 1. Each project should be aware of the performance bottlenecks in the BOINC projects if we need guaranteed performance. BOINC is geared towards large and long-term computation. The execution time of each workunit is relatively long enough, so that the BOINC server performs a relatively small amount of work distribution and reporting at the same time. Existing BOINC projects handle about $1 \sim 10$ workunits per second [12, 13].

However, in the case of computing highly-interactive and short-term tasks with deadlines, the BOINC server must perform a relatively large number of transactions per period to guarantee the worst-case performance. In Fig. 1, most of the daemon processes read/write the main database. This means that the execution time of each daemon process depends on the number of records n in the database storing application, workunit, and result data, for example. MySQL in particular has $O(\log\ n) \sim O(n^2)$ time complexity[14]. In addition, we found that daemon processes have at least linear, and up to polynomial complexity[3]. This makes it hard to provide relatively low *worst-case execution time* compared with the *average execution time* for all data-related operations and processes.

3.2 Requirements for Interactive, Short-term, and Real-time Parallel Applications

We describe the real-time requirements of the online game of Chess.

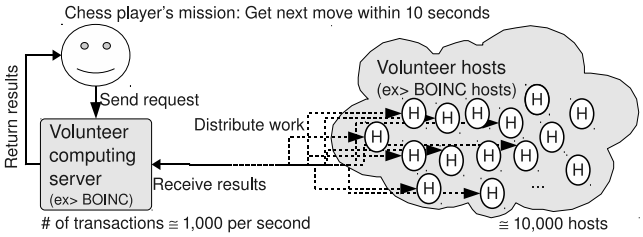

Fig. 2. An example of interactive, short-term, and real-time parallel tasks

Figure 2 shows an example calculating the next move. In this example, the Chess player wants to determine the best move by using the volunteer computing

[3] Complexities of processes; *feeder*: $O(n_w)$, *assimilator*: $O(n_w)$, *validator*: $O(n_w \cdot n_r^2)$, *transitioner*: $O(n_w \cdot n_r)$, *file-deleter*: $O(n_r)$, *work-generator*: $O(n_w)$, and *scheduler*: $O(n_r)$ where n_w is the number of workunits and n_r is the number of results.

environment. If we assume that the number of volunteer hosts is about 10,000, and each move should be calculated within *10 seconds*, then the expected number of transactions between hosts and the server is 1,000 per second. This means that the server should finish each transaction within 1 *ms*. If the applications need guarantees of real-time execution, the worst-case execution time on the server-side should be less than 1 *ms* for each transaction. To provide such a *low bounded execution time*, the internal server structures should be designed to limit the gap between the *average* and the *worst-case*.

3.3 Design of RT-BOINC

RT-BOINC was designed to provide guaranteed real-time performance for distributing work and reporting their results in the BOINC server. To do this, we modified several components of BOINC server (see Figure 3), and added new data structures and interfaces for retrieving them.

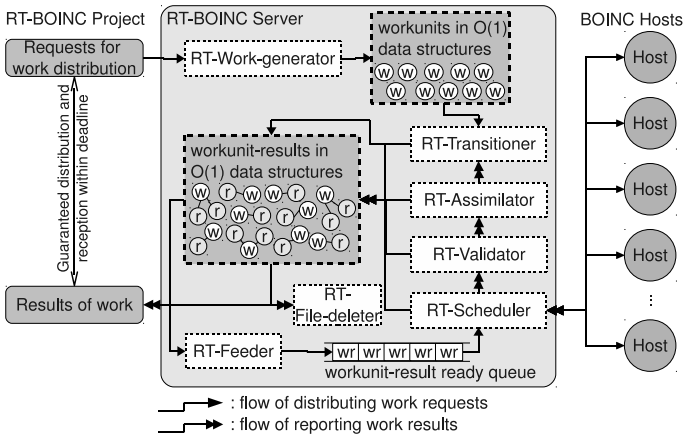

Fig. 3. Internal structures of RT-BOINC server

The major difference between the original BOINC and RT-BOINC is the management of data records. RT-BOINC does not use the database, and uses instead only *in-memory data structures* shared among daemon processes. We also modified the internal structures of the server daemon processes to reduce their complexity.

The original BOINC uses MySQL as the main DBMS, and this widens the gap between the *average* and the *worst-case* execution time for reasons discussed in Section 3.1. In RT-BOINC, we replaced the database with *in-memory data structures*, which provide $O(1)$ lookup, insertion, and deletion of data records. The data structures are shared by several daemon processes via *shared memory IPC*.

Data Structures for Real-time Operations Figure 4 shows an example of retrieving data from the shared memory data structures. In this example, we retrieve one result that has *workunitid* = 0x1234 from the result table, where *workunitid* is a field of the result. In Fig. 4, two-level lookup tables are used to reduce the *maximum length* of a list. In the worst-case, 256 entries will be scanned in search of the workunit ID.

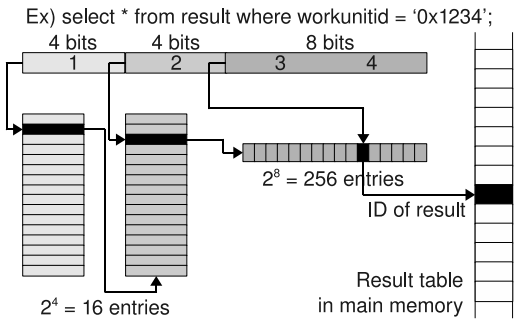

Fig. 4. An example of retrieving result in RT-BOINC

Figure 5(a) shows the case of inserting a new result to the data structures. First, we need to find a place to store the new result. To do this, we use a lookup pool for available (free) results. We can get an available result field's ID at the end of list, and remove the entry from list. Then, we can insert the result to the data structure with constant time[4]. In Fig. 5(b), we see how to delete the existing result from the data structures. If we want to delete the result which has *id* = 1234, we insert the value 1234 to the end of lookup list. Then, we can invalidate the result by removing the *valid-flag* of the result data[5].

Server Processes in RT-BOINC As we mentioned, each daemon process in BOINC has at-least linear time complexity for handling workunits and their results. To reduce complexity by orders of magnitude, we modified the internal structures of the server processes. We replaced all BOINC database-related code with $O(1)$ lookup, insertion, and deletion code. We removed unnecessary loops and redundant code from the remaining parts of the server processes.

[4] At the same time, we need to manage other lookup tables such as *workunitid*, which are presented in Fig. 4.

[5] If we have active lookup records for the result in other data structures, we should delete them also.

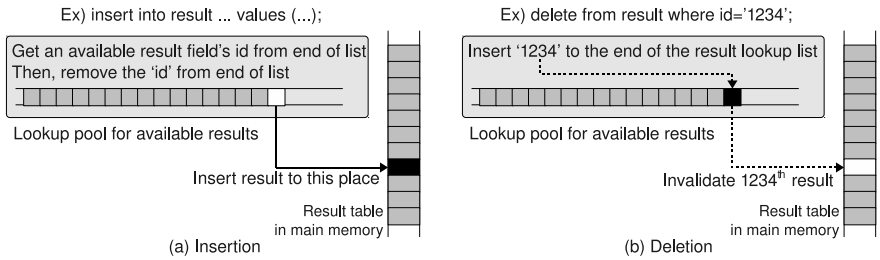

Fig. 5. Data insertion and deletion examples in RT-BOINC

3.4 Prototype Implementation

We implemented RT-BOINC on top of the BOINC server source code[6]. The full source code of RT-BOINC (prototype implementation) are available at the following website: `http://rt-boinc.sourceforge.net/`

Data Format Compaction We reduced significantly the memory consumption of the original data types of BOINC. We reduced unnecessary large blocks from *workunit*, *result*, *user*, *team*, and many other data types. For example, one *result* data has three huge fields to store XML code and standard output messages. The size of each field is 64KB, which corresponds to the MySQL BLOB data type. Then, each *result* record consumes more than 192KB, and if we have $10,000$ records on the server, this will consume more than 1.92GB of space for just handling the *result* records. However RT-BOINC does not use the database nor BLOB data types, so it does not need to consume as much space. We reduced the size of each field by a factor of 8, and we also made similar reductions for the other data types. Detailed information of data format compaction is available at the following website: `http://rt-boinc.sourceforge.net/dataformat.html`

Data Structures and Interfaces We implemented the data structures using *shared memory* IPC among several daemon processes. The prototype implementation of RT-BOINC supports up to 64K active hosts, which is reasonable based on the size of most BOINC projects [15]. To provide $O(1)$ lookup, insertion, and deletion operations on the data structures, we used two-level lookup tables and fixed-size list structures (see Figs. 4 and 5). We used a 4-bit lookup table for each level, thus each lookup table has $2^4 = 16$ fields (same as Fig. 4). Also, we made a few limitations for the prototype version of RT-BOINC. We assume that a workunit has a one-to-one relationship with a result. In our prototype, the memory space overhead for the $O(1)$ data structures is about 38.6% of the total memory usage (where the total memory requirement is 1.09GB for $10,000$ hosts).

[6] We used the *server_stable* version of BOINC in November 2009.

3.5 Compatibility with the original BOINC

All of our modifications of the BOINC server source code preserve compatibility with the original BOINC implementation and components (such as the client). In Figs. 1 and 3, we can observe that the main components of RT-BOINC are exactly the same as that of BOINC. RT-BOINC has the same set of server processes, and the flow of work distribution and reporting is the same as the original BOINC. Therefore, most of RT-BOINC server components are compatible with the existing project configurations and their applications.

4 Performance Evaluation

We made micro and macro-level benchmarks to determine the performance of both the low-level operations (such as insertion, update, and remove) and high-level server processes (such as the feeder, transitioner, and validator). We measured performance in terms of the *average*, and the *worst-case execution time* of these operations and processes in BOINC and RT-BOINC.

For the micro-benchmarks, we implemented a program that generates every possible key value for the server-side operations. Key values correspond to user, host, workunit, result id's , and other id's related to the performance.

For the macro-benchmarks, we implemented an emulator of the BOINC client, which typically runs on each volunteered host. The emulated client uses the identical protocol of the BOINC client for requesting workunits, and returning results. Moreover, it uses the server processes in the same way as the BOINC client itself. The emulated client generates server requests for all possible key values.

With regard to the workunits, we used the *uppercase* application, which is a synthetic application that converts contents of a text file to uppercase.

Table 1. Specification of the base server platform used in this paper

Component	Description	Notes
Processor	1.60GHz, 3MB L2 cache	Intel Core-2 Duo with VT
Main memory	3GB (800MHz)	Dual-channel DDR3
Secondary storage	64GB Solid State Drive	SLC type
Operating system	Ubuntu 9.10	Linux kernel 2.6.31-19

The setup in Table 1 allows us to measure the real performance in general-purpose, off-the-shelf server system.

In Fig. 6(a), Y-axis is in log-scale, and most of execution times of operations have more than a 2-step difference in terms of average execution time. In the worst-case, most times have a 3-step difference (in Fig. 6(b)). This means that RT-BOINC has improvements of more than 100-times for the average case, and

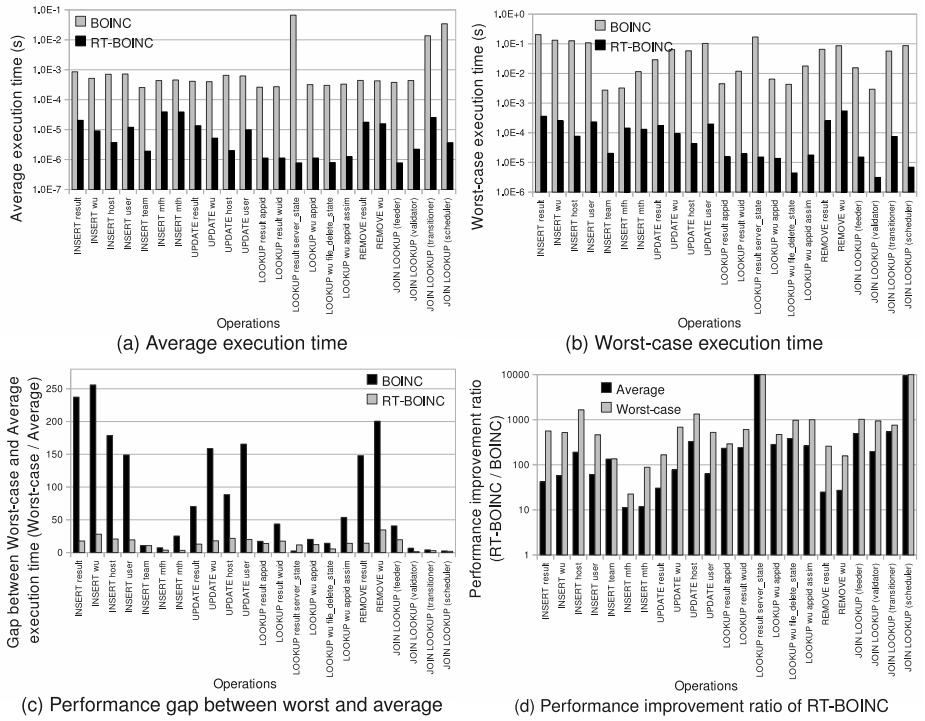

Fig. 6. Micro benchmark results of BOINC and RT-BOINC

almost 1,000-times for the worst-case (which is presented in Fig. 6(d)). In addition, Fig. 6(c) shows that RT-BOINC's performance gap between the worst and average case is much lower than that of the original BOINC. Based on these results, we can observe that RT-BOINC provides *low worst-case execution time* compared to the original BOINC for each operation.

Figure 7 shows the results of each server process in BOINC and RT-BOINC when the server has low-load conditions. In Fig. 7(a) and (c), the results show almost a 1-step difference, and Fig. 7(b) shows almost a 2-step difference between BOINC and RT-BOINC. The results show that RT-BOINC provides almost 100-times lower worst-case execution time than BOINC. In Fig. 7(d), we can observe that the performance gap of RT-BOINC is much lower than that of BOINC.

Figure 8 shows the same set of results when the server has conditions of high-load. In these results, we observe almost a 1-step difference in the average-case, and almost a 2.5-steps difference in the worst-case. Fig. 8(d) shows that BOINC has a significant difference between worst-case and average-cases, which RT-BOINC improves immensely.

From the results in Fig. 8, we found that the "end-to-end" *transaction time* of one workunit going through all server processes in RT-BOINC is 4.1 *ms* for

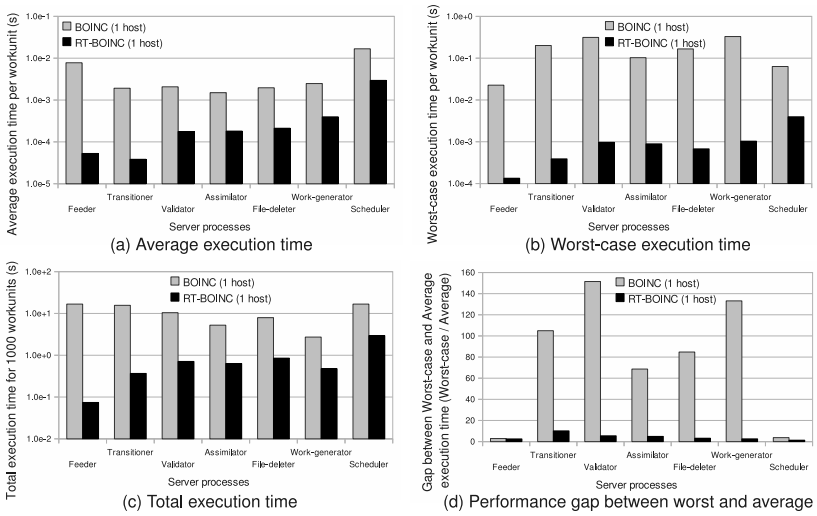

Fig. 7. Macro benchmark results of BOINC and RT-BOINC (low-load: 1 host)

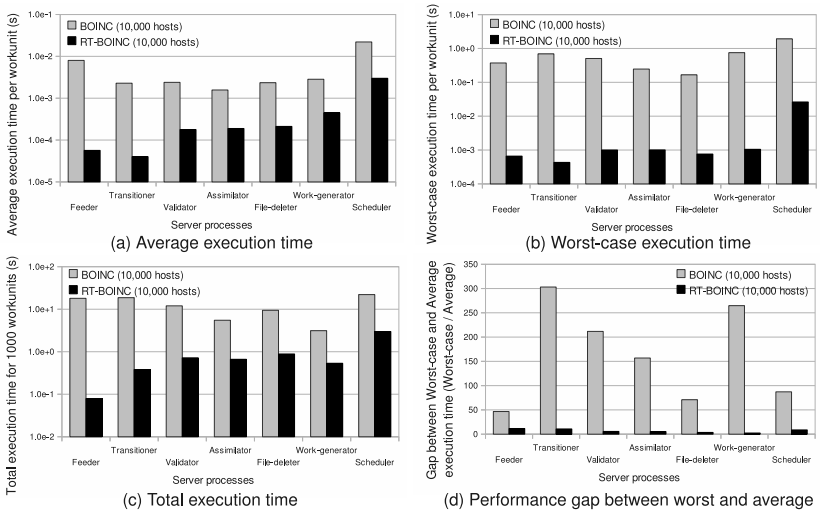

Fig. 8. Macro benchmark results of BOINC and RT-BOINC (high-load: 10,000 host)

average, and 31.2 *ms* for the worst-case *when we do not consider pipe-lined parallel execution of the server processes*. If the server can execute multiple threads at the same time, the transaction time of each workunit decreases significantly. For instance, in the average case, the server can perform 1,000 workunits per second with only 4 parallel threads of execution. The hardware setup in Table 1 supports this degree of parallelism. If the server supports 32 threads (as on a standard dual quad-core processor with hyperthreading) with the same performance presented in Fig. 8, the server can perform almost 1,000 workunits per second even in the worst-case.

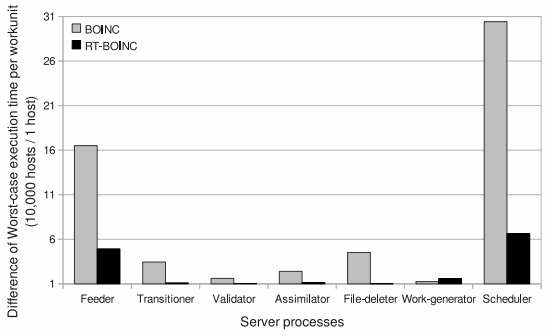

Fig. 9. Worst-case execution time difference between low and high load conditions

In Fig. 9, BOINC shows a big difference in performance than RT-BOINC under different load conditions wrt the number of hosts. This result has significant meaning on both *predictability* and *scalability* of the server system. RT-BOINC provides almost the same worst-case execution time, even if the server has significant change of host load. Based on these results, we can observe that RT-BOINC provides significantly low worst-case execution time compared with the original BOINC.

5 Conclusions and Future Work

In this paper we proposed RT-BOINC, which is a platform for real-time and highly interactive computing. This system can be used on any large distributed system, such as Clouds, Grids, or Volunteer Computing platforms, where workloads consists of tens of thousands of real-time tasks. In RT-BOINC, every component provides *bounded execution time*. Thus it helps to provide the guaranteed execution of real-time applications. We implemented it based on the original BOINC, and our evaluation results show that RT-BOINC has low *worst-case execution time* and reasonable memory space usage compared with BOINC.

For future work, we are interested in the following issues for RT-BOINC. First, we are interested in conducting dynamic shared memory management

when the server system does not have enough main-memory space. Second, we are interested in studying the trade-offs between time and space of our data structures.

Acknowledgments

This research was supported by supported the ALEAE project (INRIA ARC), and the ANR Clouds@home project (contract ANR-09-JCJC-0056-01).

References

1. Silberstein, M., Sharov, A., Geiger, D., Schuster, A.: Gridbot: Execution of bags of tasks in multiple grids. In: SC'09: Proceedings of the 2009 ACM/IEEE conference on Supercomputing, New York, NY, USA, ACM (2009)
2. Lee, C.S., Wang, M.H., Chaslot, G., Hoock, J.B., Rimmel, A., Teytaud, O., Tsai, S.R., Hsu, S.C., Hong, T.P.: The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. IEEE Transactions on Computational Intelligence and AI in games (2009)
3. Deep Blue (chess computer): http://en.wikipedia.org/wiki/Deep_Blue_(chess_computer). (website)
4. Lopez, J., Aeschlimann, M., Dinda, P., Kallivokas, L., Lowekamp, B., O'Hallaron, D.: Preliminary report on the design of a framework for distributed visualization. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99). (1999) 1833–1839
5. Capsicum Group: Digital Forensics: http://www.capsicumgroup.com/content-pages/services/digital-forensics.html. (website)
6. Dinda, P.: A Prediction-Based Real-Time Scheduling Advisor. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02). (2002)
7. Sonnek, J.D., Nathan, M., Chandra, A., Weissman, J.B.: Reputation-based scheduling on unreliable distributed infrastructures. In: ICDCS. (2006) 30
8. Andrzejak, A., Kondo, D., Anderson, D.P.: Ensuring collective availability in volatile resource pools via forecasting. In: DSOM. (2008) 149–161
9. Ratnasamy, S., Handley, M., Karp, R.M., Shenker, S.: Topologically-aware overlay construction and server selection. In: INFOCOM. (2002)
10. Kacsuk, P., Marosi, A.C., Kovacs, J., Balaton, Z., Gombs, G., Vida, G., Kornafeld, A.: Sztaki desktop grid: A hierarchical desktop grid system. In: Proceedings of the Cracow Grid Workshop 2006, Cracow (Poland) (2006)
11. Gorlatch, S., Glinka, F., Ploss, A., Müller-Iden, J., Prodan, R., Nae, V., Fahringer, T.: Enhancing grids for massively multiplayer online computer games. In: Euro-Par. (2008) 466–477
12. Catalog of BOINC Powered Projects - Unofficial BOINC Wiki: http://www.boinc-wiki.info/Catalog_of_BOINC_Powered_Projects. (website)
13. Anderson, D.P.: Talk at Condor Week, Madison, WI, http://boinc.berkeley.edu/talks/condor_boinc_06.ppt. (2006)
14. MySQL: Developer Zone: http://dev.mysql.com/. (website)
15. BOINC Statistics: http://boincstats.com/stats/. (website)