Large-Scale Software Unit Testing on the Grid

[1]Yaohang Li, [2]Tao Dong, [3]Xinyu Zhang, [4]Yongduan Song, [5]Xiaohong Yuan
[1,2,5]*Department of Computer Science*
[4]*Department of Electrical and Computer Engineering*
[3]*Department of Industrial Engineering*
North Carolina A&T State University, Greensboro, NC 27455
{yaohang, tdong, xzhang, songyd, xhyuan}@ncat.edu

Abstract

Grid computing, which is characterized by large-scale sharing and collaboration of dynamic resources, has quickly become a mainstream technology in distributed computing. In this article, we present a grid-based unit test framework, which takes advantage of the large-scale and cost-efficient computational grid resources as a software testing test bed to support automated software unit test in a complicated system. Within this test framework, a dynamic bag-of-tasks model is used to manage test suites on the grid. Moreover, an adaptive task scheduling mechanism based on swarm intelligence approach is developed to tackle the performance heterogeneity and resource dynamism problems presented in a grid-computing environment and efficiently utilize the grid resources. Overall, we expect that the grid-based unit test framework can significantly reduce test cost in complex software systems and accelerate the testing process with large number of unit test suites.

1. Introduction

Revolutionary changes in software engineering during last several decades have already become a commonplace. With the development of modern methods, technologies, and tools, the functionality and complexity of modern systems grow exponentially. However, behind the evolution of software there has been one big problem – quality control techniques lag behind the development ones and cannot provide the same level of quality for complex systems with manageable growth of effort. Software testing is an integral, costly, and time-consuming activity in the software development life cycle. The ISSRE'97 panel [1] stated the following facets of large-scale software testing.

> *"Large software product organizations spend 50% or more of their budgets on testing. Testers comprise 20% to 50% of software personnel in many companies. With all this effort, best in class software products still may contain 400 faults per million lines of fielded code. The cost of repairing field failures is growing rather than shrinking."*

As a result, reducing software testing cost will directly lead to significant overall software development cost reduction.

Within all software testing levels, unit test is the fundamental one, which goes together to make the "big picture" of testing a system. In software engineering literature [15], a unit is defined as the smallest collection of code which can be usefully tested. Typically, a unit would be a non-trivial object class, a subroutine, a script, or a module of source code. A unit test is a procedure used to verify whether a particular unit is working correctly or not. The main idea about unit tests is to write test cases for all units so that whenever a change causes a regression, it can be quickly identified and fixed. Ideally, each test case is separate from the others, constructing mock objects that can assist in separating unit tests. A software system with non-trivial complexity is usually composed of large number of units while each unit may have test cases ranging from several to thousands or even more. Every test case must be executed many times along the software development life cycle when a new module is added, an existing functionality is modified, or a software defect is fixed. As a result, in a large and complex software system, running large number of unit test suites is rather computationally costly. Therefore, a powerful test bed with large-scale of computational capability, which can automatically and effectively carry out unit test suites, is needed for complicated software system testing.

Grid computing is characterized by large-scale sharing and cooperation of dynamically distributed resources, such as CPU cycles, communication bandwidth, and data, to constitute a computational environment [2, 3]. A computational grid based on the grid-computing techniques can, in principle, provide tremendously large amount of low-cost CPU cycles, which may be utilized for speeding up software testing [4]. Moreover, software portability testing on different system configurations is always a formidable task within the software testing process. A computational grid is a heterogeneous computing environment, which is usually comprised of a large array of hardware architecture, operating system, and middleware library combinations. All these make a computational grid a natural and possibly an ideal testbed for large-scale software testing.

In this article, we try to address a question – can the grid dramatically improve the state-of-practice in large-scale software testing? Despite the attractive characteristics of grid computing, successfully exploiting the grid techniques for large-scale software testing depends on overcoming a number of challenges stemming from properties of a computational grid such as

resource dynamism, performance heterogeneity, trustworthiness, and cross-domain physicality [2, 5, 23]. We developed a grid-based software testing framework to facilitate the automated process of utilizing the grid resources for software unit testing. Within this testing framework, we propose a swarm intelligence scheduling strategy to improve grid resources utilization efficiency and reduce testing task completion time. Overall, we expect that our grid-based unit test framework can speedup large-scale software testing process and reduce testing cost.

The remainder of this paper is organized as follows. In Section 2, we analyze the characteristics of grid-computing environment and issues for utilizing the grid for software testing. We describe our implementation of the grid-based software testing framework and its underlying dynamic bag-of-tasks model in Section 3 and Section 4, respectively. We present an innovative swarm intelligence testing task scheduling mechanism in Section 5. Finally, Section 6 summarizes our conclusions and future research directions.

2. Issues of Software Testing on a Grid

Recently, grid computing has emerged as an important new area in parallel computing, distinguished from traditional distributed computing by its focus on large-scale dynamic, distributed, and heterogeneous resource sharing, cooperation of organizations, innovative applications, and high-performance orientation. Many applications have been developed to take advantage of grid computing facilities and have already achieved elementary success. However, as the field grew so also did the problems associated with it. Traditional assumptions that are more or less valid in traditional distributed and parallel computing settings break down on the Grid. In traditional distributed computing settings, one often assumes a "well-behaved" system: no faults or failures, minimal security requirements, consistency of state among application components, availability of global information, and simple resource sharing policies. While these assumptions are arguably valid in tightly coupled systems, they break down as systems become much more widely distributed [7]. First of all, the grid is a dynamic computing environment without centralized control. Nodes providing computational services join and leave dynamically. Secondly, nodes in a computational grid exhibit heterogeneous performances. The capabilities of each node vary greatly. A node might be a high-end supercomputer, or a low-end personal computer, even just an intelligent widget. As a result, a task running on different nodes on the grid will have a huge range of completion times. In [4], Durate et al. showed that the instability of network significantly affects the performance of distributed software testing. Thirdly, there may be untrustworthy nodes existing in the grid. Therefore, all these issues mentioned above must be addressed before a large-scale computational grid can be efficiently utilized to speedup large-scale software testing.

3. Grid-based Unit Test Framework

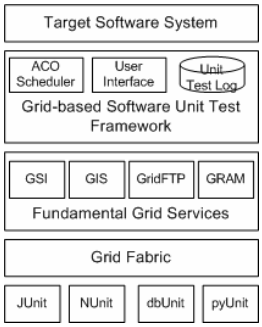

Figure 1: System Architecture of Grid-based Software Testing Framework

The grid-based software unit test framework is designed on top of the common grid services provided by the Globus toolkit [6, 18], including GRAM (Globus Resource Allocation Manager), GIS (Grid Information Service), GSI (Grid Security Infrastructure), and GridFTP. GRAM is used to submit testing tasks to grid nodes and to manage the execution of the test cases. GIS provides information services. GSI offers security services such as authentication, encryption, and decryption for running testing tasks on the grid. GridFTP provides a uniform interface for transporting unit test suites and test results between grid service providers. Via the fundamental grid services provided by Globus, the grid-based unit test framework can invoke the local unit test tools, such as JUnit [18], NUnit [19], and dbUnit [20], according to the programming language requirement of the target software system, to carry out various unit test suites. Moreover, the framework employs a test task scheduler based on Ant Colony Optimization (ACO) and provides user interface and test suites logging for target software systems. Figure 1 illustrates the overall system architecture of the grid-based unit test framework.

4. Dynamic Bag-of-Tasks Model

The grid-based unit test framework manages test suites using the dynamic bag-of-tasks (also called bag-of-work) model [8, 9, 10], which applies to the situation when embarrassingly parallel computational tasks are to be executed a large number of times. The dynamic bag-of-tasks computing paradigm favors applications with embarrassingly parallel characteristics, i.e., situations where the overall testing scenario can be easily divided into smaller independent testing tasks. This situation arises naturally in large-scale unit test scenario, where every test case is independent. A set of parameters for a unit test suite constitutes a testing task, and the collection of all tasks to be executed is called the bag of tasks, since

they do not need to be solved in any particular order. A testing task may contain thousands of test cases and may come to the bag of tasks at any time. Also, to improve reliability of the software testing, replicated testing tasks may present in the bag of tasks [21].

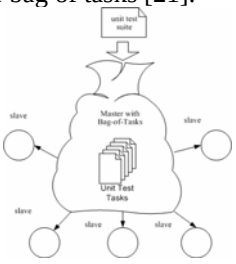

Figure 2: The Dynamic Bag-of-Tasks Paradigm

The dynamic bag-of-tasks model usually employs the master-slave scheduling paradigm, where the master is responsible of dispatching appropriate tasks while multiple slaves carry out their tasks. When a slave is free, the master will select an appropriate testing task from the bag and assign it to the slave. After successfully receiving the task, the slave then computes a partial test result. Finally, the master collects the distributed partial test results to generate a testing report. Figure 2 shows the master-slave scheduling paradigm in the dynamic bag-of tasks model for large-scale unit tests.

5. ACO for Testing Tasks Scheduling

The goal of task-scheduling algorithm of the grid-based unit test framework is to minimize the execution time of the unit test tasks by efficiently taking advantage of the large amount of distributed computational resources available in the grid. In our implementation of the grid-based unit test framework, a swarm intelligent scheduling approach based on Ant Colony Optimization (ACO) [16, 17] is employed to tackle the performance heterogeneity and resource dynamism problems presented in the grid.

In our scheduling mechanism using swarm intelligence, we design various software ant agents, including scouts, workers, collectors, cleaners, and queens, with simple functionalities. No direct communications occur among these ants. The only indirect communication is via the pheromone values stored in a grid resource table. The swarm intelligence mechanism of testing task scheduling on the computational grid is depicted as follows:

1. Initially, the queen spawns scouts, cleaners, and workers. The queen also produces testers at a time period of T.
2. A scout visits the information services providers of the grid and explores those nodes providing computational services. The scout finds the available nodes and adds them to the grid resource table with initial pheromone value, θ.
3. Once a testing task is submitted to the computational grid, a worker will try to schedule this task to an available node providing computational service. A node having a higher pheromone value will be selected with a higher probability. A node i will be selected with probability, q_i, of $q_i = p_i / \sum_{j=1}^{n} p_j$, where p_i is the pheromone value of node i and n is the total number of available nodes in the grid computing environment.
4. Global and local pheromone updates are presented in the algorithm. In local update, when a test task is complete on a grid node, the collector will retrieve the test result and update the pheromone value $p_i \leftarrow p_i + \Delta p_i$, where Δp_i is the newly added pheromone amount. Global updates take place at every period of time T_1. The grid node who completes most schedule units will obtain an extra bonus update of its pheromone value $p_i \leftarrow p_i + \rho$, where ρ is the bonus pheromone value. The global update increases the chance of a fast node to be selected and thus accelerates the convergence to the optimal grid path.
5. The pheromone values of nodes evaporate. At every period of time T_2, the pheromone value of every grid node is updated as $p_i \leftarrow \gamma p_i$, where $\gamma < 1$ is the evaporation constant.
6. When the pheromone value of a node is lower than some threshold value, τ, which indicates that this node has been unavailable for a long time or is an extremely slow node with an undesired task completion time, the cleaner will remove it from the grid resource table.

In this swarm intelligence scheduling algorithm, variables T_1, T_2, θ, γ, and τ, are tunable parameters subject to the specific grid computing environment.

We compare the swarm intelligence scheduling approach with two widely used scheduling approaches, random scheduling and heuristic scheduling [22]. Random scheduling approach has no extra information of the performance of a rid node and, thus, schedules testing tasks to its grid nodes in a random manner. Heuristic scheduling approach considers the previous overall performance of a grid node in managing workload on different grid nodes. Figure 3 illustrates the testing task completion times verse task arrival rates on our simulated computational grid. At each time step, the performance of every node within the simulated grid changes with a probability of 0.0001. The swarm intelligence mechanism shows a better average task completion time than the random mechanism and the heuristic mechanism. Figure 4 depicts the adaptability of the swarm intelligence scheduling mechanism. As the grid node performance changing probability increases, the simulated grid with fixed task arrival rate evolves from a slightly dynamic system to a heavily dynamic system. The performances of

the heuristic mechanism and the random mechanism change dramatically; in contrast, the swarm intelligence mechanism exhibits a rather steady task completion time and yields almost the best task completion time in all these situations.

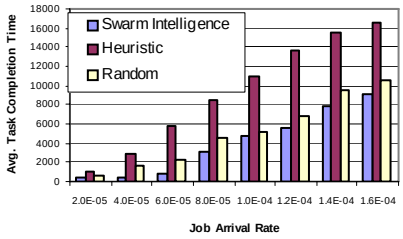

Figure 3: Swarm intelligence mechanism, heuristic mechanism, and random mechanism on a simulated computational grid

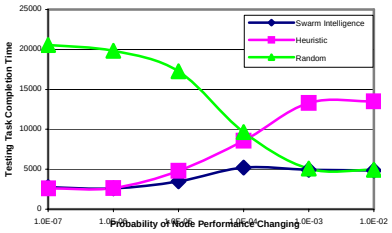

Figure 4: Performance comparison of swarm intelligence mechanism, heuristic mechanism, and random mechanism on a simulated computational grid with different probabilities of node performance changing

6. Conclusions and Future Research Directions

In this paper, we presented the implementation of a grid-based unit test framework to take advantage of the large-scale and cost-efficient computational grid resources to build a testbed to speedup software testing process and reduce testing cost. Also, we proposed a swarm intelligence task scheduling approach to tackle the dynamism and performance heterogeneity problems in grid computing environment with expectation of reduction of testing task completion time. Our preliminary simulation results confirm our expectation.

Currently, our grid-based unit test framework can only run on several Linux computers and our presented preliminary results are based on simulation are just for "proof-of-concept." At the next step, we must verify the practical feasibility of the framework. We plan to expand the test framework to the existing large-scale grid test bed, such as the NC Grid [21] or GridLab [22].

Acknowledgement

This work is partially supported by the "Building an NCA&T Campus Grid Project" of UNC General Administration and the NC-HPC Project of UNC President's Office.

References

[1] R. Horgan, "Panel Statement: Large Scale Software Testing," Proc. of 8th International Symposium on Software Reliability Engineering, 1998.

[2] I. Foster, C. Kesselman, and S. Tueske, "The Anatomy of the Grid," International Journal of Supercomputer Applications, 15(3), 2001.

[3] C. Goble, D. D. Roure, "The Grid: An Application of the Semantic Web," Grid Computing: Making the Global Infrastructure a Reality, pp. 437-470, 2003.

[4] A. N. Duarte, W. Cirne, F. Brasileiro, P. Duarte, L. Machado, Using the Computational Grid to Speed up Software Testing, Proc. of 19th Brazilian Symp. on Software Engineering, 2005.

[5] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, "The Physiology of Grid: Open Grid Services Architecture for Distributed Systems Integration," draft, 2003.

[6] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," International Journal of Supercomputer Applications, 11(2), 1997.

[7] H. Casanova, Distributed Computing Research Issues for Grid Computing, ACM SIGACT, 33(2), 2002.

[8] Carriero, N., Gelernter, D., Leichter, J., Distributed Data Structures in Linda, Proc. of 13th ACM Symp. on Principles of Programming Languages, 1986.

[9] Andrews, G. R., Concurrent Programming: Principles and Practice, Benjamin/Cummings, 1991.

[10] Kuang, H., Bic, L. F., Dillencourt, M. B., Iterative Grid-Based Computing Using Mobile Agents, Proceedings of 31st IEEE ICPP2002, 2002.

[11] R. Pressman, Software Engineering: A Practitioner's Approach, McGraw Hill, 2005.

[12] E. Bonabeau, G. Théraulaz, Swarm Smarts,"Scientific American, pp. 72-79, 2000.

[13] A. Colorni, M. Dorigo, V. Maniezzo, Distributed Optimization by Ant Colonies, Proc. of 1st European Conference on Artificial Life, 1992.

[14] JUnit, http://www.junit.org/index.htm, 2005.

[15] NUnit, http://www.nunit.org, 2005.

[16] DbUnit, http://dbunit.sourceforge.net, 2005.

[17] Globus Toolkit, http://www.globus.org, 2005.

[18] M. Wu, X. Sun, A General Self-adaptive Task Scheduling System for Nondedicated Hetero-Computing, Proc. of IEEE Int. Conf. on Cluster Comp., 2003.

[19] NC Grid, http://www.mcnc.org/, 2005.

[20] GridLab, http://www.gridlab.org/, 2005.

[21] L. F. G. Sarmenta, Sabotage-Tolerance Mechanisms for Volunteer Computing Systems, Proc. of ACM/IEEE CCGrid'01, 2001.

[22] Y. Li, M. Mascagni, Grid-based Monte Carlo Applications, Proc. of the International Conference on Grid Computing, GRID2002, Baltimore, 2002.