

Achieving Fault Tolerance in Grid Computing System

Meenakshi Bheevgade, Manik Mujumdar, Dr. Rajendra Patrikar, Latesh Malik

Abstract— Grid computing is a means of allocating the computational power of a large number of computers to complex difficult computation or problem. Grid computing is a distributed computing paradigm that differs from traditional distributed computing in that it is aimed toward large scale systems that even span organizational boundaries. We propose a novel method to achieve maximum fault tolerance in the Grid environment by using Reliability consideration.

I. INTRODUCTION

Computational grids have become a popular approach to handle vast amounts of available information and to manage computational resources. Examples of areas where grids have been successfully used for solving problems include biology, nuclear physics and engineering.

A Grid enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources.

Stand-alone system is prone to crash. These individual components in a distributed computing system may fail without stopping the entire computing system. So Fault tolerance is an important property in Grid computing as grid resources are geographically distributed in different administrative domains worldwide. Also in large-scale grids, the probability of a failure is much greater than in traditional parallel systems.

Since grid applications run in a very heterogeneous computing environment, fault tolerance is important in order to ensure their correct behavior. The development of correct grid applications is difficult with traditional software development methods. Hence, formal methods can be beneficial in order to ensure their correctness and structure their development from specification to implementation.

One of the most difficult tasks in the design of a fault-tolerant Grid environment is to verify that it will meet its reliability requirements. Performance models for two fault tolerance methods, checkpoint-recovery (CR) and wide-area replication (WR), have been considered.

Application checkpoint-restart is the ability to save the state of a running application to secondary storage so that it can later resume its execution from the time at which it was last stored. Checkpoint-restart can provide many potential benefits, including

Fault recovery by rolling back an application to a previous checkpoint, Better application response time by restarting applications from checkpoints instead of from

scratch, and improved system utilization by stopping long running computationally intensive jobs during execution and restarting them when load decreases.

An application can be migrated by check pointing it on one machine and restarting it on another providing further benefits, including fault resilience by migrating applications from the faulty hosts, dynamic load balancing by migrating applications to less loaded hosts, and improved service availability and administration by migrating applications before host maintenance so that applications can continue to run with minimal downtime.

Many important applications consist of multiple co-operating processes. To checkpoint-restart these applications, not only must application state associated with each process be saved and restored, but the state saved and restored must be globally consistent and preserve process dependencies. Furthermore, for checkpoint restart to be useful in practice, it is crucial that it transparently support the large existing installed base of applications running on commodity operating systems.

The ability to checkpoint a running application and restart it later can provide many useful benefits including fault recovery, advanced resources sharing, dynamic load balancing and improved service availability. However, applications often involve multiple processes which have dependencies through the operating system.

A common method of ensuring the progress of a long running application is to take a checkpoint i.e., save its state on stable storage periodically. A checkpoint is an insurance policy against failures—in the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be recomputed.

The state of a distributed application consists of the instantaneous snapshot of the local state of processes and communication channels. However, in an asynchronous distributed system with no global clocks or shared memory, we can only devise algorithms to approximate this global state [7]. A snapshot is deemed consistent if it could have occurred during the execution of an application [7, 4]. To yield a consistent snapshot, or checkpoint, an algorithm must ensure that all messages received by a process are recorded as having been sent.

II PROPOSED WORK

We present a transparent mechanism for commodity operating systems that can checkpoint multiple processes in a consistent state so that they can be restarted correctly at a later time. We introduce an efficient algorithm for recording process relationships at system-level with checkpoint/restart implementation for Linux clusters that targets typical High Performance Computing applications, including MPI. Application process state including shared resources and various identifiers that define process relationships such as group and session identifiers are saved and restored correctly. The process will be migrated from a failed node to a spare node instead of restarting the application using checkpoint policy.

We try to achieve the maximum fault tolerance by using Reliability consideration. Reliability defects generally are failures that might occur in the future inside a system that has been working well so far. Therefore, reliability must be regarded as a ratio expressed in terms of units of time.

If a system is well designed and carefully configured, and thus homogeneously reliable, the accumulated failure rate of the system, $F(t)$, is proportional to the time period of acceptable life to job assigned to the node, including on and off time[5]

$$F(t)=1-R(t)=1-e^{-\lambda t} \approx \lambda t$$

where R is reliability function, λ is failure rate, and t is time requirement of the job. By using the above formula we can calculate the number of spare nodes based on the Failure Rate.

REFERENCES

- [1] Oren Laadan Jason Nieh, Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems, 2007 USENIX Annual Technical Conference
- [2] J. H. Abawajy, Fault-Tolerant Scheduling Policy for Grid Computing Systems, 2004 IEEE
- [3] Babar Nazir, Taimoor Khan, Fault Tolerant Job Scheduling in Computational Grid, 2006 IEEE, pp 708-713
- [4] Mattern, F., Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation, Journal of Parallel and Distributed Computing, pp. 423-434, 1993.
- [5] D. Ryu, Quality, product quality, and market share increase, Int J Reliab Appl 2 (2001) (3), p. 163 174,182.
- [6] Paul Stelling, Ian Foster, Carl Kesselman, Craig Lee, Gregor von Laszewski, A Fault Detection Service for Wide Area Distributed Computations
- [7] Chandy, K. M., Lamport, L., Distributed Snapshots: Determining Global States of Distributed Systems, ACM Transactions on Computer Systems, pp. 63-75, February 1985.
- [8] I. Foster, C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. The

International Journal of Supercomputer Applications,15(3), 2001.

[9] I. Foster, C. Kesselman, J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[10] URL:

<http://www.globus.org/alliance/publications/papers/ogsa.pdf>

[11] Jin Liang, Tong WeiQin, Tang JianQuan, Wang Bo, A Fault Tolerance Mechanism in Grid, 0-7803-5/04/2003, IEEE pp. 457-461

[12] J. H. Abawajy and S. P. Dandamudi. Parallel job scheduling on multi-cluster computing systems. In Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2003), Hong Kong, China, December 1-4 2003.

[13] A. Avizienis, "The N-version Approach to Fault-Tolerant Software" - IEEE Transactions on Software Engineering - vol. 11 1985

[14] T. Thanalapati and S. Dandamudi. An efficient adaptive scheduling scheme for distributed memory multicomputers. IEEE Transactions on Parallel and Distributed Systems, 12(7):758-768, July 2001.