Comparative study of Fault Tolerance techniques

Minakshi Memoria
Department of Computer Science and Engineering
World College of Technology and Management, Gurgaon, India

Dr. Ripu Ranjan Sinha
Department of Research and Development
Gyan Vihar University, Jaipur, India

Dr. Mukesh Yadav
Department of Computer Science and Engineering
Gurgaon Institute of Technology and Management,Gurgaon, India

Abstract- In this paper, Grid computing is the collection of computer resources from different locations to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grid computing is distinguished from conventional high performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application. This paper investigates the intrinsic difference between different systems that are used for the implementation of fault detection and tolerance.

Keywords – Cuckoo search, Genetic algorithm, lazy classifier and neural network classifier, Email spam

I. INTRODUCTION

Distributed systems and Grid Systems are well known for achieving high performance in computing. We distribute all jobs into portions and send them to the machines for computations that are part of the distributed system. Nodes that are part of the distributed system execute their portion of the job and submit the results to job submission node. Distributed systems are further classified into Clusters and Grids. If we want to establish a reliable and available distributed environment then a fault-tolerant mechanism should be there. The incorporation of faults handling mechanisms in Clusters and Grids play an important role for that environment to be reliable and available. Fault tolerance is a capability developed in the system so that it could perform its function correctly even in the presence of faults. Taking fault tolerance into consideration would result in increasing the dependability of a system [15]. According to [15] failure is encountered when a system moves away from its particular behavior. The reason behind that failure is called error that also ultimately depicts some sort of fault or defect in that system. This means that fault is the actual cause of a failure, and error is just an indication or sign of a fault. Multiple errors could be due to a fault, and even a single error could be the cause of multiple failures. In fault tolerance we try to preserve the delivery of expected services in the presence of faults that can cause errors. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service [16].

II. EXISTING FAULT TOLERANCE TECHNIQUES

Many fault tolerance techniques such as retry, replication, message logging and check pointing [20] are available in traditional distributed paradigms.

i. Retry

Retry is the simplest failure recovery technique in which we hope that whatever is the cause of failures, the effect will not be encountered in subsequent retries [17].

ii. Replication

In replication based technique we have replicas of a task running on different machines and as long as not all replicated tasks crash (i.e. host crash etc.), chances are that the task execution would succeed [17].

iii. Message Logging

In message logging all participating nodes log incoming messages to stable storage and when a failure is encountered than these message logs are used to compute a consistent global state. Algorithms that take this approach can be further classified into those that use pessimistic and those that use optimistic message logging [18].

iv. Check-pointing

Check-pointing is relatively more popular fault tolerant approach used in distributed systems, where the state of the application is stored periodically on reliable and stable storage, normally a hard disk etc. In case any fault occur during execution, i.e. after crash etc., the application is restarted from the last checkpoint rather than from the beginning [19].

v. Evaluation

For performing critical evaluation, we have compared different Grid Fault Tolerance implementations with each other and found some advantages and disadvantages of those techniques. Same technique was performed for Clustered based Fault Tolerant implementation scenarios.Table1 shows their differences.

TABLE 1 summarizes fault detection and tolerance techniques used in parallel and distributed system

System	Type	Fault detection technique	Fault tolerance technique	Comments
Globus[1]	**Grid**	**Heart beat monitor**	**Resubmit the failed job**	**Can't handle user defined exceptions. It provides uniform and secure environment for accessing remote computational and storage resources.**
LA-MPI[2]	**Cluster**	**Checks unacknowledged list at specific intervals**	**Sender side retransmission**	**Appropriate only for low error rate environments. It can't do process migration.**
LAM/MPI+BLCR[4]	**Cluster**	**Node/Application stops responding**	**Replication of checkpoints**	**Communications increases by replication checkpoints on several machines. It can do process migration also.**
Legion[5]	**Grid**	**Pinging and Timeout**	**Checkpoint recovery**	**Can't distinguish between task crash failure and host/network failure**
NetSolve [6]	**Grid**	**Generic heart beat mechanism**	**Retry on another available machine**	**Doesn't support diverse failure recovery mechanism. A programming and runtime system for accessing high performance libraries and resource transparently.**
Nimrod-G[8]	**Grid**	**Uses deadline constraint**	**Four strategies of DBC scheduling algorithm**	**An economic based Grid resource broker for parameter sweep/task framing applications**

A. Globus

Globus[2] provides a software infrastructure that enables applications to handle distributed heterogeneous computing resources as a single virtual machine. A computational Grid, in this context, is a hardware and software infrastructure that provides dependable, consistent and pervasive access to high end computational capabilities,

despite the geographical distribution of both resources and users. It provides basic services and capabilities that are required to construct a computational Grid. The toolkit consists of a set of components that implement basic services, such as security, resource location, resource management, and communications.

It is necessary for computational Grids to support a wide variety of applications and programming paradigm. Consequently, rather than providing a uniform programming model, such as the object oriented model. It provides lot of services which developers of particular tools or applications can use to meet their own particular needs. This methodology is only possible when the services are distinct and have well defined interfaces that can be incorporated into applications or tools in incremental fashion.

The Globus supports the following:

Grid security infrastructure

GridFTP

Globus resource allocation manager

Metacomputing directory service

Global access to secondary storage

Data catalogue and replica management

Advanced resource reservation and allocation

B. LA-MPI

LA-MPI[3] is an implementation of MPI in which we address fault tolerance at all of these levels. It Reliably delivers messages in the presence of I/O bus, network card and wire transmission errors Survives network card and path failures and guarantees delivery of inflight messages after such a failure Supports the concurrent use multiple types of network interface and Implements message striping of message fragments across multiple homogeneous network interfaces.

There have been a number of research efforts attempting to incorporate network and process fault tolerance into message passing system. It follows checkpoint/ rollback recovery system. It provides end to end reliability in a h of high performance message passing system without significant overhead on a wide variety of network transports and devices.

It also offers the possibility to enhance performance relative to existing message passing systems by implementing message striping across multiple heterogeneous network interfaces, and message fragment striping across multiple homogeneous network interfaces. It can't do process migration.

C. LAM-MPI+BLCR

Instead of job restart, LAM-MPI[4] is a transparent mechanism for job pause which allows live nodes to remain active and roll back to the last checkpoint while failed nodes are dynamically replaced by spared before resuming from the last checkpoint. This includes LAM/MPI enhancements in support of scalable group communication with fluctuating number of nodes, reuse of network connections, transparent coordinated checkpoint scheduling and a BLCR enhancement of job pause.

The mechanism, implemented within LAM/MPI+BLCR, allows live nodes to remain active and roll back to the last checkpoint while failed nodes are dynamically replaced by spares before resuming from the last checkpoint. Enhancements to LAM/MPI include

i. Support of scalable group communication with fluctuating number of nodes,

ii. Transparent coordinated checkpointing,

iii. Reuse of network connections upon failures for operational nodes, and

iv. a BLCR enhancement for the job pause mechanism.

We have conducted experiments with the NAS Parallel Benchmark suite in a 16-node dual-processor Opteron cluster. Results indicate that the performance of job pause is comparable to that of a complete job restart, albeit at full transparency and automation. A minimal overhead of 5.6% is only incurred in case migration takes place while the regular checkpoint overhead remains unchanged. Yet, our approach alleviates the need to reboot the LAM run-time

environment, which accounts for considerable overhead resulting in net savings of our scheme in the experiments. Furthermore, job pause reuses existing resources and continues to run within the scheduled job, which can avoid staging overhead and lengthy requeuing in submission queues associated with traditional job restarts. Our experiments also indicate that, after the initialization phase, checkpoints are constant in size for a given application, regardless of the timing of checkpoints.

D. Legion

Legion[5] is an object-based metasystem developed at the University of Virginia. Legion provides the software infrastructure so that a system of heterogeneous, geographically distributed, high-performance machines can interact seamlessly. Legion attempts to provide users, at their workstations, with a single, coherent, virtual machine. In the Legion system the following apply.

- Everything is an object. Objects represent all hardware and software components. Each object is an active process that responds to method invocations from other objects within the system. Legion defines an API for object interaction, but not the programming language or communication protocol.

- Classes manage their instances.Every Legion object is defined and managed by its own active class object. Class objects are given system-level capabilities; they can create new instances, schedule them for execution, activate or deactivate an object, as well as provide state information to client objects.

Users can define their own classes.As in other object-oriented systems users can override or redefine the functionality of a class. This feature allows functionality to be added or removed to meet a user's needs. Legion core objects support the basic services needed by the metasystem. The Legion system supports the following set of core object types.

- Classes and metaclasses. Classes can be considered managers and policy makers. Metaclasses are classes of classes.

- Host objects. Host objects are abstractions of processing resources, they may represent a single processor or multiple hosts and processors.

- Vault objects.Vault objects represent persistent storage, but only for the purpose of maintaining the state of Object Persistent Representation (OPR).

- Implementation objects and caches. Implementation objects hide the storage details of object implementations and can be thought of as equivalent to executable files in UNIX. Implementation cache objects provide objects with a cache of frequently used data.

- Binding agents. A binding agent maps object IDs to physical addresses. Binding agents can cache bindings and organize themselves into hierarchies and software combining trees.

- Context objects and context spaces. Context objects map context names to Legion object IDs, allowing users to name objects with arbitrary-length string names. Context spaces consist of directed graphs of context objects that name and organize information.

Legion objects are independent, active, and capable of communicating with each other via unordered non-blocking calls. Like other object-oriented systems, the set of methods of an object describes its interface. The Legion interfaces are described in an Interface Definition Language (IDL). The Legion system uses an object-oriented approach, which potentially makes it ideal for designing and implementing complex distributed computing environments. However, using an object-oriented methodology does not come without a raft of problems, many of these being tied-up with the need for Legion to interact with legacy applications and services.

E. Netsolve

NetSolve[7] is a client/server application designed to solve computational science problems in a distributed environment. The Netsolve system is based around loosely coupled distributed systems, connected via a LAN or WAN. Netsolve clients can be written in C and Fortran, and use Matlab or the Web to interact with the server. A Netsolve server can use any scientific package to provide its computational software. Communications within Netsolve is via sockets. Good performance is ensured by a load-balancing policy that enables NetSolve to use the computational resources available as efficiently as possible. NetSolve offers the ability to search for computational resources on a network, choose the best one available, solve a problem (with retry for fault-tolerance), and return the answer to the user.

F. Nimrod-G

Nimrod-G is a Grid resource broker that performs resource management and scheduling of parameter sweep, task-farming applications on worldwide Grid resources [8,9]. It consists of four key components: a task-farming engine, a scheduler, a dispatcher, and agents (see Figure 4 for the Nimrod-G broker architecture). A Nimrod-G persistent and programmable task-farming engine (TFE) enables 'plugging' of user-defined schedulers and customized applications or problem-solving environments (e.g. ActiveSheets [10]) in place of default components. The dispatcher uses the Globus services to deploy Nimrod-G agents on remote resources in order to manage the execution of assigned jobs. The local resource management system (e.g. queuing system or forking service) starts the execution of the Nimrod-G agent that interacts with the I/O server running on the user home/root-node to fetch a task script assigned to it (by the Nimrod-G scheduler) and executes the Nimrod commands specified in the script. The Nimrod-G scheduler has the ability to lease Grid resources and services depending on their capability, cost, and availability driven by user QoS requirements. It supports resource discovery, selection, scheduling, and transparent execution of user jobs on remote resources. The users can set the deadline by which the results are needed; the Nimrod/G broker then tries to find the cheapest computational resources available on the Grid and use them so that the user deadline is met and the cost of computation is kept to a minimum.

Specifically, Nimrod-G supports user-defined deadline and budget constraints for schedule optimizations and manages the supply and demand of resources in the Grid using a set of distributed computational economy and resource trading services called GRACE (Grid Architecture for Computational Economy) [11]. The deadline and budget constrained (DBC) scheduling algorithms with four different optimization strategies [12,13]—cost optimization, cost-time optimization, time optimization, and conservative-time optimization—supported by the Nimrod-G resource broker for scheduling applications on the worldwide distributed resources are shown in Table VII. The cost optimization scheduling algorithm uses the cheapest resources to ensure that the deadline can be met and the computational cost is minimized. The time optimization scheduling algorithm uses all the affordable resources to process jobs in parallel as early as possible. The cost-time optimization scheduling is similar to cost optimization, but if there are multiple resources with the same cost, it applies the time optimization strategy while scheduling jobs on them. The conservative time optimization scheduling strategy is similar to the time-optimization scheduling strategy, but it guarantees that each unprocessed job has a minimum budget-per-job. The Nimrod-G broker with these scheduling strategies has been used to solve large-scale data-intensive computing applications such as the simulation of ionization chamber calibration [8] and the molecular modeling for drug design [14].

IV.CONCLUSION

Specifically, this is comparative study of six systems that are implemented for fault detection and tolerance in distributed and grid systems. Globus, LA-MPI, LAM/MPI+BLCR, Legion, NetSolve and Nimrod-G in which four are grid type and two are cluster. All are using different fault detection and tolerance techniques, criteria and method.

REFERENCES

[1] Mark Baker, RajkumarBuyya and DomenicoLaforenza, "Grids and Grid Technologies for Wide-Area Distributed Computing", A technical report August, 2002

[2] Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. International Journal of Supercomputer Applications 1997; 11(2):115–128.

[3] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalski. "A network-failure-tolerant message passing system for terascale clusters". In Proceedings of the 16th international conference on Supercomputing, pages 77–83. ACM Press, 2002.

[4] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "A Job Pause Service under LAM/MPI+BLCR for Transparent Fault Tolerance," in IPDPS '07: Proceedings of the 21st International Parallel and Distributed Processing Symposium. IEEE Computer, 2007, pp. 116–125.

[5] S. Chakravorty and C. Mendes and L. V. Kal´e, "Proactive Fault Tolerance in MPI Applications via Task Migration," in HiPC '06: Proceedings of the 13th International Conference on High Performance Computing, LNCS 4297, 2006, pp. 485–496.

[6] Soon Hwang and Carl Kesselman "A Flexible Framework for Fault Tolerance in the Grid", Journal of Grid Computing 1: 251–272, 2003.

[7] Casanova H, Dongarra J. NetSolve: A network server for solving computational science problems. International Journal of Supercomputing Applications and High Performance Computing 1997; 11(3).

[8] Abramson D, Giddy J, Kotler L. High performance parametric modeling with Nimrod/G: Killer application for the global Grid? International Parallel and Distributed Processing Symposium (IPDPS). IEEE Computer Society Press: Los Alamitos, CA, 2000.

[9] Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia'2000), Beijing, China, 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.

[10] Abramson D, Roe P, Kotler L, Mather D. ActiveSheets: Super-computing with spreadsheets. 2001 High Performance Computing Symposium (HPC'01), Advanced Simulation Technologies Conference, April 2001. SCS Press: San Diego, CA, 2001.

[11] Buyya R, Abramson D, Giddy J. Economy driven resource management architecture for computational power grids. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, NV, 2000. CSREA Press: Athens, GA, 2000.

[12] Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. The Second Workshop on Active Middleware Services (AMS 2000), in Conjunction with HPDC 2001, 1 August 2000, Pittsburgh, PA. Kluwer Academic Press, 2000.

[13] Buyya R, Murshed M, Abramson D. A deadline and budget constrained cost-time optimization algorithm for scheduling task farming applications on global Grids. Technical Report, Monash University, March 2002. http://www.buyya.com/gridsim/.

[14] Buyya R. The Virtual Laboratory Project: Molecular modeling for drug design on Grid. IEEE Distributed Systems Online 2001; 2(5). http://www.buyya.com/vlab/.

[15] Bran Selic, "Fault tolerance techniques for distributed systems", Staff, IBM, Software Group, 27th July 2004.

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

[17] Soon Hwang and Carl Kesselman "A Flexible Framework for Fault Tolerance in the Grid", Journal of Grid Computing 1: 251–272, 2003.

[18] A. P. Sistla , J. L. Welch, Efficient distributed recovery using message logging, Proceedings of the eighth annual ACM Symposium on Principles of distributed computing, p.223-238, June 1989, Edmonton, Alberta, Canada.

[19] N Hussain, M. A. Ansari, M. M. Yasin, A Rauf, S Haider, Fault Tolerance using "Parallel Shadow Image Servers (PSIS)" in Grid Based Computing Environment, IEEE—ICET, 13-14 November 2006.

[20] PankajJalote, "Fault Tolerance in Distributed Systems", ISBN: 0-13-301367-7, 1994.