Rollback Recovery Through Check Points in Heterogeneous Grid Computing

P.Venkata Maheswara
Assistant Professor, Dept. Of CSE,
KMM Inistitute of Technology and Sciences
Tirupati,India

Ch. Chengamma
Assistant Professor, Dept. Of CSE,
KMM Inistitute of Technology and Sciences
Tirupati,India

***Abstract*:-Large applications executing on Grid or cluster architectures consisting of hundreds or thousands of computational nodes create problems with respect to reliability. The sources of the problems are node failures and the need for dynamic configuration over extensive runtime. This paper presents two fault-tolerance mechanisms called Theft-Induced Check pointing and Systematic Event Logging. These are transparent protocols capable of overcoming problems associated with both benign faults, i.e., crash faults, and node or subnet volatility.**

Specifically, the protocols base the state of the execution on a dataflow graph, allowing for efficient recovery in dynamic heterogeneous systems as well as multithreaded applications. By allowing recovery even under different numbers of processors, the approaches are especially suitable for applications with a need for adaptive or reactionary configuration control. The low-cost protocols offer the capability of controlling or bounding the overhead. A formal cost model is presented, followed by an experimental evaluation.

I. INTRODUCTION

GRID and cluster architectures have gained popularity for computationally intensive parallel applications. However, the complexity of the infrastructure, consisting of computational nodes, mass storage, and interconnection networks, poses great challenges with respect to overall system reliability. Simple tools of reliability analysis show that as the complexity of the system increases, its reliability, and thus, Mean Time to Failure (MTTF), decreases. The reliability of the entire system is computed as the product of the reliabilities of all system components. The high failure probabilities are due to the fact that, in the absence of fault-tolerance mechanisms, the failure of a single node will cause the entire execution to fail. Note that this simple example does not even consider network failures, which are typically more likely than computer failure. Fault tolerance is, thus, a necessity to avoid failure in large applications, such as found in scientific computing, executing on a Grid, or large cluster.

The fault-tolerance mechanisms also have to be capable of dealing with the specific characteristics of a heterogeneous and dynamic environment. Even if individual clusters are homogeneous, heterogeneity in a Grid is mostly unavoidable, since different participating clusters often use diverse hardware or software architectures. One possible solution to address heterogeneity is to use platform independent abstractions such as the Java Virtual Machine. However, this does not solve the problem in general.

There is a large base of existing applications that have been developed in other languages. Reengineering may not be feasible due to performance or cost reasons. Environments like Microsoft .Net address portability but only few scientific applications on Grids or clusters exist. Whereas Grids and clusters are dominated by Unix operating systems, e.g., Linux or Solaris, Microsoft .Net is Windows-centric with only recent or partial

Unix support. Besides heterogeneity, one has to address the dynamic nature of the Grid. Volatility is not only an intracluster issue, i.e., configuration changes within a cluster, but also an intercluster reality.

Intracluster volatility may be the result of node failures, whereas intercluster volatility is caused by network disruptions between clusters. From an administrative viewpoint, the reality of Grid operation, such as cluster/node reservations or maintenance, may restrict long executions on fixed topologies due to the fact that operation at different sites may be hard to coordinate. It is usually difficult to reserve a large cluster for long executions, let alone scheduling extensive uninterrupted time on multiple, perhaps geographically dispersed, sites. Lastly, configuration changes may be induced by the application as the result of changes of runtime observable quality-of-service (Quos) parameters. To overcome the aforementioned problems and challenges, we present mechanisms that tolerate faults Operation-induced disruption of parts or the entire execution of the application. We introduce flexible rollback recovery mechanisms that impose no artificial restrictions on the execution. They do not depend on the refailure configuration and consider

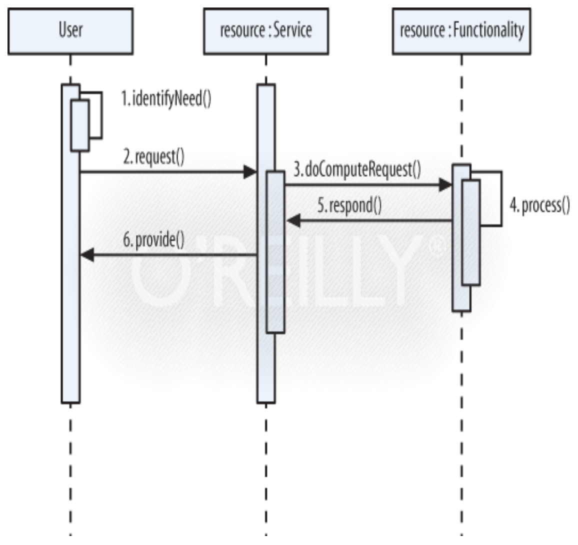
1) Node and cluster failures as well as operation-induced unavailability of resources and

2) Dynamic topology reconfiguration in heterogeneous systems.

II. MODULE DESCRIPTION

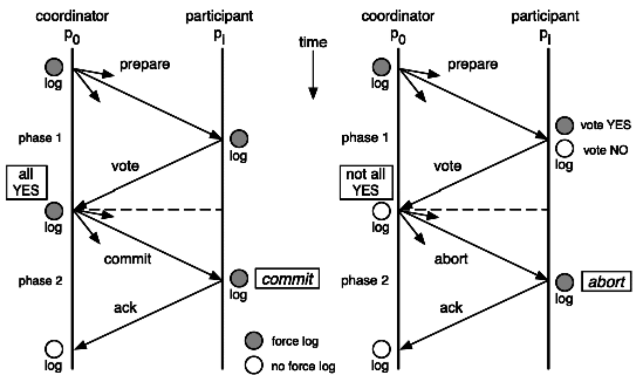
1. Network Module:

Client-server computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client also shares any of its resources; Clients therefore initiate communication sessions with servers which await (listen to) incoming requests.



2. *Logging Module:*

Logging can be classified as pessimistic, optimistic, or causal. It is based on the fact that the execution of a process can be modeled as a sequence of state intervals. The execution during a state interval is deterministic. However, each state interval is initiated by a nondeterministic event. Now, assume 0that the system can capture and log sufficient information about the nondeterministic events that initiated the state interval. This is called the piecewise deterministic (PWD) assumption .Then, a crashed process can be recovered by 1) restoring it to the initial state and 2) replaying the logged events to it in the same order they appeared in the execution before the crash. To avoid a rollback to the initial state of a process and to limit the amount of nondeterministic events that need to be replayed, each process periodically saves its local state. Log-based mechanisms in which the only nondeterministic events in a system are the reception of messages is usually referred to as message logging.

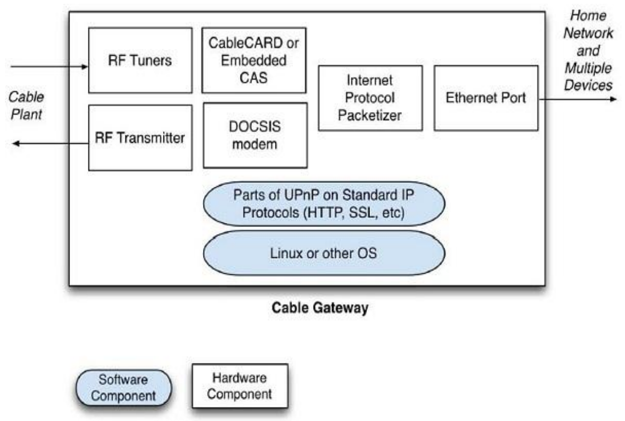


3. *Check-pointing Module*

Rather than logging events, check pointing relies on periodically saving the state of the computation to stable storage. If a fault occurs, the computation is restarted from one of the previously saved states. Since the computation is distributed, one has to consider the tradeoff space of local and global check pointing strategies and their resulting recovery cost. Thus, check pointing based methods differs in the way processes are coordinated and in the derivation of a consistent global state. The consistent global state can be achieved either at the time of check pointing or at the time of rollback recovery. The two approaches are called coordinated and uncoordinated check pointing, respectively.

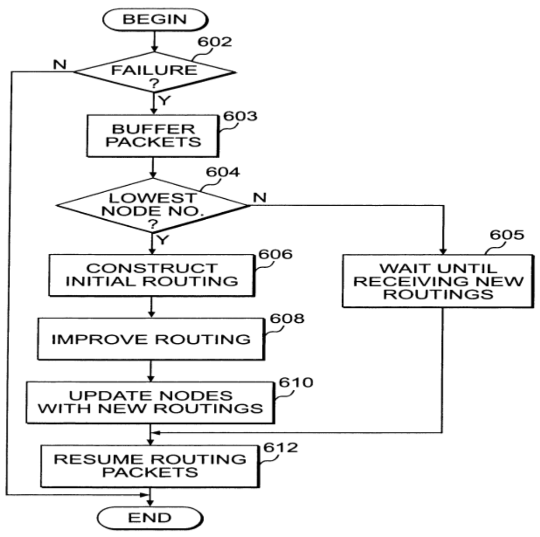
4. *Work Stealing Module:*

The runtime environment and primary mechanism for load distribution is based on a scheduling algorithm called work-stealing .The principle is simple: when a process becomes idle it tries to steal work from another process called victim. The initiating process is called thief. Work-stealing is the only mechanism for distributing the workload constituting the application, i.e., an idle process seeks to steal work from another process. From a practical point of view, the application starts with the process executing main (), which creates tasks. Typically, some of these tasks are then stolen by idle processes, which are either local or on other processors. Thus, the principal mechanism for dispatching tasks in the distributed environment is task stealing


Cable Plant
RF Tuners
CableCARD or Embedded CAS
Internet Protocol Packetizer
Ethernet Port
Home Network and Multiple Devices
RF Transmitter
DOCSIS modem
Parts of UPnP on Standard IP Protocols (HTTP, SSL, etc)
Linux or other OS
Cable Gateway
Software Component
Hardware Component

5. *Fault and Fault Free Module***:**

We add a check pointing mechanism; it is of special interest to analyze its overhead associated with fault-free execution, since the occurrence of faults is considered to be the rare exception rather than the norm.



III. IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system and is giving confidence on the new system for the users, which it will work efficiently and effectively. It involves careful planning, investigation of the current System and its constraints on implementation, design of methods to achieve the change over, an evaluation, of change over methods. Apart from planning major task of preparing the implementation are education and training of users. The more complex system being implemented, the more involved will be the system analysis and the design effort required just for implementation.

An implementation co-ordination committee based on policies of individual organization has been appointed. The implementation process begins with preparing a plan for the implementation of the system. According to this plan, the activities are to be carried out, discussions made regarding the equipment and resources and the additional equipment has to be acquired to implement the new system. Implementation is the final and important phase, the most critical stage in achieving a successful new system and in giving the users confidence. That the new system will work be effective.

User Training:

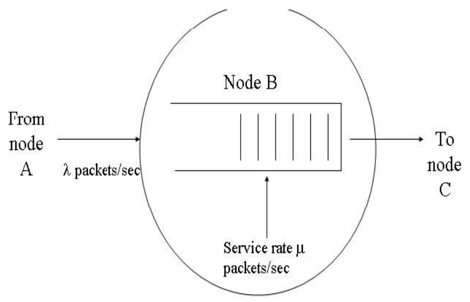
After the system is implemented successfully, training of the user is one of the most important subtasks of the developer. For this purpose user manuals are prepared and handled over to the user to operate the developed system. Thus the users are trained to operate the developed systems successfully in future .In order to put new application system into use, the following activities were taken care of:

- Preparation of user and system documentation
- Conducting user training with demo and hands on
- Test run for some period to ensure smooth switching over the system

The users are trained to use the newly developed functions. User manuals describing the procedures for using the functions listed on menu and circulated to all the users .it is confirmed that the system is implemented up to user need and expectations.

Protocol :

It detects traffic faulty routers by validating the queue of each output interface for each router. Given the buffer size and the rate at which traffic enters and exits a queue, the behavior of the queue is deterministic. If the actual behavior deviates from the predicted behavior, then a failure has occurred. We present the failure detection protocol in terms of the solutions of the distinct sub problems: traffic validation, distributed detection, and response.



IV. CONCLUISON

To overcome the problem of applications executing in large Systems where the MTTF approaches or sinks below the execution time of the application, two fault-tolerant protocols, TIC and SEL, were introduced. The two protocols take under consideration the heterogeneous and dynamic characteristics of Grid or cluster applications that pose limitations on the effective exploitation of the underlying infrastructure. The flexibility of dataflow graphs has been exploited to allow for a platform-independent description of the execution state. This description resulted in flexible and portable rollback recovery strategies. SEL allowed for rollback at the lowest level of granularity, with a maximal computational loss of one task. However, its overhead was sensitive to the size of the associated dataflow graph. TIC experienced lower overhead, related to work-stealing, which was shown bounded by the critical path of the graph. By selecting an appropriate application granularity for SEL and period _ for TIC, the protocols can be tuned to the specific requirements or needs of the application. A cost model was derived, quantifying the induced overhead of both protocols. The experimental Results confirmed the theoretical analysis and demonstrated the low overhead of both approaches. ACKNOWLEDGMENTS the authors wish to thank Jean-Louis Roche, ID-IMAG, France, for all the discussions and valuable insight that led to the success of this research.

V. REFERENCES

1. Bouteiller et al., "MPICH-V2: A Fault Tolerant MPI for Volatile Nodes Based on the Pessimistic Sender Based Message Logging," Proc. ACM/IEEE Conf. Supercomputing (SC '03), pp. 1-17, 2003.
2. Bouteiller, P. Lemarinier, G. Krawezik, and F. Cappello, "Coordinated Checkpoint versus Message Log for Fault Tolerant MPI," Proc. Fifth IEEE Int'l Conf. Cluster Computing (Cluster '03), p. 242, 2003.
3. S. Chakravorty and L.V. Kale, "A Fault Tolerant Protocol for Massively Parallel Machines," Proc. 18th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS '04), p. 212a, 2004.
4. K.M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," ACM Trans. Computer Systems, vol. 3, no. 1, pp. 63-75, 1985.
5. E.N. Elnozahy, L. Alvisi, Y.-M. Wang, and D.B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys, vol. 34, no. 3, pp. 375-408, Sept. 2002.
6. S. Jafar, T. Gautier, A. Krings, and J.-L. Roch, "A Checkpoint/ Recovery Model for Heterogeneous Dataflow Computations Using Work-Stealing," Proc. European Conf. Parallel Processing (EuroPar '05), pp. 675-684, Aug.-Sept. 2005.
7. G. Zheng, L. Shi, and L.V. Kale´, "FTC Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI," Proc. Sixth IEEE Int'l Conf. Cluster Computing (Cluster '04), pp. 93-103, Sept. 2004.
8. A.W. Krings, J.-L. Roch, S. Jafar, and S. Varrette, "A Probabilistic Approach for Task and Result Certification of Large-Scale Distributed Applications in Hostile Environments," Proc. European Grid Conf. (EGC '05), P. Sloot et al., eds., Feb. 2005.