ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com

International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

An Adaptive Framework for Reliable Distributed Computing

R.SENTHIL KUMAR[1], S.VIVEK PANDIAN[2], S.ABIRAMI[3],

Assistant Professor, Department of Computer Science Engineering,
R.V.S Educational Trust's Group of Institution's, Dindigul, India[123]

***Abstract*-Most application level fault tolerance schemes in existing systems are non-adaptive in the sense that the fault tolerance schemes incorporated in applications are usually designed without incorporating information from system environments such as the amount of available memory and the local or network I/O bandwidth. However, from an application point of view, it is often desirable for fault tolerant high performance applications to be able to achieve high performance under whatever system environment it executes with as low fault tolerance overhead as possible. Here we demonstrate that, in order to achieve high reliability with as low performance penalty as possible, fault tolerant schemes in applications need to be able to adapt themselves to different system environments. An effective adaptive fault tolerance system is required for real time application. An adaptive fault tolerance system does the following activities. If three systems are running simultaneously, job is submitted to all system. If one system fails, our adaptive framework will check the load conditions of running system and it will submit to the least loaded system**.

***Index Terms*-Check points, fault tolerance, failure recovery, SMP, snapshots**

I. INTRODUCTION

Fault-tolerance or graceful degradation is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. If its operating quality decreases at all, the decrease is proportional to the severity of the failure, as compared to a naïvely-designed system in which even a small failure can cause total breakdown. Fault-tolerance is particularly sought-after in high-availability or life-critical systems.

Fault-tolerance is not just a property of individual machines; it may also characterize the rules by which they interact. For example, the Transmission Control Protocol (TCP) is designed to allow reliable two-way communication in a packet-switched network, even in the presence of communications links which are imperfect or overloaded.

It does this by requiring the end points of the communication to expect packet loss, duplication, reordering and corruption, so that these conditions do not damage data integrity, and only reduce throughput by a proportional amount.

A. Fault tolerance requirements

The basic characteristics of fault tolerance require:

- No single point of failure
- No single point of repair
- Fault isolation to the failing component
- Fault containment to prevent propagation of the failure
- Availability of reversion modes

Fault-tolerant systems are typically based on the concept of redundancy.

B. No single point of repair

If a system experiences a failure, it must continue to operate without interruption during the repair process.

C. Fault isolation to the failing component

When a failure occurs, the system must be able to isolate the failure to the offending component. This requires the addition of dedicated failure detection mechanisms that exist only for the purpose of fault isolation. Recovery from a fault condition requires classifying the fault or failing component.

Due to the large process state of such kind of applications, the relatively low I/O bandwidth between memory and the central network disk, and the high enough frequency of failures, for these systems, the classical system-level fault tolerance approaches is often either impractical (an application would spend most of its time taking checkpoints) or infeasible (there is no enough time for an application to save its core to disk before the next failure occurs). Therefore the cheaper application level fault tolerance schemes may be deployed as an alternative in such large computational science programs.

However, most application level fault tolerance schemes proposed in literature are non-adaptive in the sense that the fault tolerance schemes incorporated in applications are either designed without incorporating system environments(such as the amount of available memory and the local and network I/O bandwidth,etc) or designed only for a specific system environment. From the application point of view, fault tolerant high performance applications need to be able to achieve high performance under different system environments with as low performance overhead as possible. In order to achieve high reliability and survivability with low performance overhead, the fault tolerance schemes in such applications need to be adaptable to different (or dynamic)system environments. In this paper, we propose a framework under which different failures can be incorporated in applications using an adaptive method. In our framework, applications will be able to choose the best available fault tolerance at runtime (or dynamically) according to different (or dynamic) system environments.

II. FAULT TOLERANCE IN PARALLEL AND DISTRIBUTED SYSTEMS

In recent years, Multi-Processors (MP), Symmetrical Multi-Processors (SMP), and Massively Parallel Processors (MPP) have been sweeping the marketplace and gaining ground to offload vast amounts of data processing. This processing is performed in "parallel" among the available Central Processing Units (CPUs).

Parallel processing is an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process. Concurrent implies parallelism, simultaneity , and pipelining .Parallel events may occurs in multiple resources during the same time interval, simultaneous events may occurs at the same time instant, and pipelined events may occur in overlapped time spans. Parallel processing demands concurrent execution of many programs in the computer. Parallel processing and distributed processing are closely related each other. In some cases, we use certain distributed technique to achieve parallelism.

Fault tolerance techniques can be divided into two big branches and some hybrid techniques. The first branch is Messaging Logging. In this branch, there are three sub branches: Pessimistic Messaging Logging, Optimistic Messaging Logging., and Casual Messaging Logging. The second branch is Check pointing and Rollback recovery. There are also three sub-branches in this branch: Network disk based Check pointing and rollback recovery, Diskless Check pointing, and Local Disk based check pointing.

Our research is mainly concentrated on incorporating failures into tightly coupled large scale high performance computational intensive applications. These applications are often communication intensive, so checkpoint and rollback recovery approaches generally work better than message logging approaches. In the rest of this section, we confine our literature review to check pointing and rollback recovery schemes instead of general fault tolerance schemes
Most traditional distributed multiprocessor recovery schemes are designed to tolerant arbitrary number of failures. So they store their checkpoint data in a central stable storage. The central stable storage usually has its own fault tolerance techniques to prevent it from failures. But the bandwidth between the processors and the central stable storage is usually very low. Several experimental studies presented in have shown that the main performance overhead of check pointing is the time spent on writing the checkpoint data to the central stable storage.
In summary, a review of the existing fault tolerance research demonstrates that

- To tolerate arbitrary number of failures with low performance overhead, a two-level (or multi-level) recovery scheme should be used.
- If enough memory is available, Checkpoint Mirroring should be used rather than Parity Based Check pointing.
- If there is no enough memory but there is enough local disk storage available, local disk storage can be use to reduce the checkpoint performance overhead.
- To achieve low performance overhead, user defined check pointing schemes should be used instead of the system-level check pointing schemes.

III. MOTIVATIONS FOR AN ADAPTIVE FAULT TOLERANCE

We have seen that the previous fault tolerant research works have produced some very precious result. However, there appears to be a significant gap between the fault tolerant research results and their optimal deployment into applications. Each fault tolerance scheme has its own advantages and disadvantages. Different systems have different resource characteristics. From the application point of view, it is desirable that fault tolerant high performance applications is able to achieve both high performance and high reliability (survivability)with low fault tolerance overhead no matter under which kind of system environments it is running. To achieve this goal, the best strategy would be to adaptively choose the fault tolerance schemes in applications based on different (or dynamic) system environments that the applications are running.

A. Reasons for using fault tolerant algorithms

Increasing the number of components in a distributed system means increasing the probability that some of these components will be subject to failure during the execution of a distributed algorithm. Computers in a network may fail, processes in a System can be erroneously killed by switching off a workstation, or a machine may produce an incorrect result due to, memory malfunctioning. Modern computers are becoming more and more in any individual computer. Nonetheless, the chance of a failure occurring at some place in a distributed system may grow arbitrarily large when the algorithm each time a failure occurs, algorithms should be designed so as to deal properly with such failures.

Vulnerability to failures is also a concern in sequential computations, in safety-critical application, or if a computation runs for a long time and produces a non-verifiable result. Internal checks protect against errors of some types but of course no protection can be achieved against the complete loss of the program or erroneous changes in its code. Therefore the possibilities of fault-tolerant computing by sequential algorithms and uniprocessor computing systems are limited. Fortunately, the study of fault-tolerant algorithms has advanced considerably since 1981, and reliable applications based on replication are now well within reach. In robust algorithms each step of each process is taken with sufficient care to ensure that, in spite of failures, correct processes only take correct steps. In stabilizing algorithms correct processes can be affected by

ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com
International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

failures, but the algorithm is guaranteed to recover from any arbitrary configuration when the processes resume correct behavior.

B. Failure models

To determine how the correctly operating processes can protect themselves against failed processes, assumptions must be made about how a process might fail. In the following chapters it is always assumed that only processes can fail; channels are reliable. Thus, if a correct process sends a message to another correct process, receipt of the message within finite time is guaranteed. (A failing channel can be modeled by a failure in one of the incident processes, for example, an omission failure.) As an additional assumption, we always assume that each process can send to each other process. The fault models are:

Initially dead processes: A process is called initially dead if it does not execute a single step of its local algorithm.

Crash model: A process is said to crash if it executes its local algorithm correctly up to some moment, and does not execute any step thereafter.

Byzantine behavior: A process is said to be Byzantine if it executes arbitrary steps that are not in accordance with its local algorithm. N particular, a Byzantine process may send messages with an arbitrary content.

C. Failure detection

The impossibility of solving consensus in asynchronous systems has led to weaker problem formulations and stronger models. Failure detectors are new widely recognized as an alternative way to strengthen the computation model.Studying synchronous models is practically motivated because most distributed programming environments do provide clocks and timers in some way. With failure detectors, the situation is similar; quite often the run-time support system will return error messages upon an attempt to communicate with a crashed process. However, these error messages are not always absolutely reliable. It is therefore useful to study how reliable they must be to allow a solution for the consensus problem.

IV. AN ADAPTIVE APPLICATION LEVEL FAULT TOLERANCE SCHEME

In this section, we present an adapting application level fault tolerance scheme for high performance grid computing.

A. Overview

Our goal is to establish a framework under which different failures can be optimally incorporated in applications using an adaptive method. In our framework, applications will be able to adaptively choose the best (minimizing the mean execution time of the application) available fault tolerance at runtime according to different system environments. Different fault tolerant schemes require different resources. When designing the fault tolerant application, the application developer may not have apriority knowledge of the system characteristics of the platform the application will be running on. Therefore, an adapting application level fault tolerance scheme need to be able to detect system information at run time. The system characteristics that is necessary in determining checkpoint schemes may include

- The number of available processors
- The amount of available memory on each processor
- The amount of available local disk storage on each processor
- Whether there is a central fail free stable storage available

Different fault tolerant schemes have different degree of reliability. To tolerate the failure of all processors, a central stable storage is usually necessary. However, if we want to tolerate only a small number of processor failures, a central stable storage is usually not necessary. For example, schemes such as neighbor-based diskless check pointing also means that both its memory and its local disk become work fine to tolerate single processor failure. In order to maximize the degree of reliability while maintaining low performance overhead, a multi-level recovery scheme is often desirable in an adapting application level fault tolerance scheme.

B. A Multi-level adaptive recovery scheme

Assume a processor can access the following five types of storage in the computing system

- local memory of the processor
- local disk of the processor
- neighbor processors' memory
- neighbor processors' disk
- central stable storage

If one type of storage is not available in the system, then we assume there are zero bytes of that type of storage in the system. We also assume that the bandwidth of these five types' storages is strictly decreasing. Assume a node failure unavailable. Which kind of checkpoint schemes (or combination, or modification of schemes) is best for a specific system is affected by many factors. At the present time, we only consider the following factors:

- The amount of available storage of each kind
- The overhead of each checkpoint scheme (which is mainly dependent on the bandwidth of each storage and the characteristics of that checkpoint schemes)
- The failure distribution of the system.
- The characteristics of the application
- The number of available processors for this application.

The five candidate basic checkpoint schemes that we consider at the present time are

- CSSC: Central Stable Storage Checkpoint scheme
- NDPC: Neighbor Disk-based Parity Checkpoint scheme
- NDCM: Neighbor Disk-based Checkpoint Mirroring scheme
- NMPC: Neighbor Memory-based Parity checkpoint scheme
- NMCM: Neighbor Memory-based Checkpoint Mirroring scheme

The multi-level adaptive recovery scheme is the combination

ISSN (ONLINE) : 2395-695X
ISSN (PRINT) : 2395-695X
Available online at www.ijarbest.com

International Journal of Advanced Research in Biology, Ecology, Science and Technology (IJARBEST)
Vol. 1, Issue 9, December 2015

of some of the above five basic schemes. Just as shown in existing research works, on most systems, the performance of these five basic recovery schemes is increasing (but it is also possible in the future to perform experiments to decide the performance of different schemes at run time).Since we also know the degree of fault tolerance of each basic scheme, so which combination to choose is mainly dependent on the availability and the amount of each storage. The checkpoint frequency of each basic scheme is mainly decided by the overhead of the scheme and the failure rate of the system.

V. EXPERIMENTAL RESULTS

We evaluate the performance of the proposed adapting fault tolerance scheme experimentally.

A. *Snapshots*

An algorithm whose task is to analyze properties of computations, usually arising from other algorithms. It is, however, surprisingly hard to observe the computations of a distributed system from within the same system. An important building block in the design of algorithms operating on system computations is a procedure for computing and storing a single configuration of this computation called snapshot. The construction of snapshots is motivated by several applications, of which we list three here. First, properties of the computation, as far as they are reflected within a single configuration, can be analyzed off-line, i.e., by an algorithm that inspects the (fixed) snapshot rather than the (varying) actual process states. These properties include stable properties; a property P of configurations is stable if P(r) r-> =>P (d). If a computation ever reaches a configuration r for which P holds, P remains true in every configuration d from then on. Consequently, if P is found to be true for a snapshot of the configuration, the truth of P can be concluded for all configurations from then on. Examples of stable properties include termination, deadlock, loss of tokens, and non-reach ability of objects in dynamic memory structure. Second, a snapshot can be used instead of the initial configuration if the computation must be restarted due to a process failure. To this end, the local state Cp for process P, captured in the snapshot, is restored in that process, after which the operation of the algorithm is continued. Third, snapshots are a useful tool in debugging distributed programs. An off-line analysis of a configuration taken from an erroneous execution may reveal why a program does not act as expected.

B. *Distributed computing environment*

The performance of the DCE is entirely depends on number of participants included in a particular process using multithreading. Where number of participants increase, system failure rate is also increased to make the DCE reliable, fault detection, fault tolerance and failure recovery must be done.

C. *Fault tolerance*

While a participant fail, leader check whether the job to be continue or redo leader increment the failure system count and check with the initial total number of participants.

"t" – number of faulty system "n" –
number of participants
if 2t < n then the current process can be continue otherwise the entire process must be redo.

D. *Failure recovery*

If failure system is less then total number of participants then failure recovery can be done. Recovery can be done by system properties stored in the adjacent system. The performance depends on number of agents included in a process. Total time taken to execute a job is equal to division of time taken by single CPU and number of participants plus communication delay.

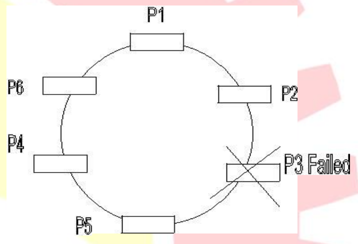

Fig 1: Failure recovery

P1, P2, P3, P4, P5, P6 – Processing systems (Participants)
P3 job can be takeover by P4 or P2

VI. CONCLUSION

Mainframes and Parallel computers are highly reliable and cost number crunches. The recent decade of client server technology evolution indicates cost as a prime factor. There by, identifying distributed object based solutions and Linux clusters. With full advantage of economy distributed object based solution still stays as a bumbling amateur. As the first pedestal, we had designed a Reliable Distributed Computing Environment in windows platform. Current implementation of the distributed computing environment is designed using Message based Middleware this can be enhanced with naming service. Intelligence can be added to enhance the job distribution. Current implementation of the distributed computing environment is designed to run in Microsoft Network under windows-NT architecture (Intranet), this can be extended to Internet with the help of WEB SERVERS.

REFERENCES

[1] N. R. Adiga and et al. An overview of the BlueGene/L supercomputer. In Proceedings of the Supercomputing Conference (SC'2002), Baltimore MD, USA, pages 1– 22, 2002.

[2] N. H. Vaidya. A case for two-level recovery schemes.IEEE Trans. Computers, 47(6):656–666, 1998.

[3] L. M. Silva and J. G. Silva.An experimental study about diskless check pointing. In EUROMICRO'98,pages 395–402, 1998.

[4] T. Chiueh and P. Deng. Evaluation of checkpoint mechanisms for massively parallel machines. In FTCS, pages 370–379, 1996.

[5] J. Dongarra, H. Meuer, and E. Strohmaier. TOP500 Supercomputer Sites, 28th edition. In Proceedings of the Supercomputing Conference (SC'2006), Pittsburgh PA, USA. ACM, 2006.

[6] G. E. Fagg and J. Dongarra. FT-MPI: Fault tolerant MPI, supporting dynamic applications in a dynamic world. In PVM/MPI 2000, pages 346–353, 2000.

[7] G. E. Fagg, E. Gabriel, G. Bosilca, T. Angskun,Z. Chen, J. Computing Infrastructure. Morgan Kauffman, San Francisco, 1999.

[8] Y. Kim. Fault Tolerant Matrix Operations for Parallel and Distributed Systems. Ph.D. dissertation, University of Tennessee, Knoxville, June 1996.

[9] Pjesivac-Grbovic, K. London, and J. J.Dongarra. Extending the MPI specification for process fault tolerance on high performance computing systems. In Proceedings of the International Supercomputer Conference, Heidelberg, Germany, 2004.

[10] G. E. Fagg, E. Gabriel, Z. Chen, , T. Angskun,G. Bosilca, J. Pjesivac-Grbovic, and J. J. Dongarra. Process fault-tolerance: Semantics, design and applications for high performance computing. Submitted to International Journal of High Performance Computing Applications, 2004.

Mr. R.Senthil Kumar was born on 8th July1987. He did his B.E – CSE in PSNA College of Engineering and Technology, Anna University in 2008 and M.E (Computer Science Engineering) in PSNA College of Engineering and Technology, in the year 2010. He is currently working as Assistant Professor in Department of Computer Science Engineering in RVS Educational Trust's Group of Institutions. His area of specialization is Distributed Computing and published papers in several National and International Conferences and Journals

Mr. S.Vivekpandian was born on 1st may1987. He did his B.E – CSE in Trichy Engineering College, Anna University in 2010 and M.E (Computer Science Engineering) in MAM College of Engineering in the year 2012. He is currently working as Assistant Professor in Department of Computer Science Engineering in RVS Educational Trust's Group of Institutions. His area of specialization is Cloud Computing and published papers in several National and International Conferences and Journals

Mrs. S.Abirami was born on 11th June1984. She did his B.E – CSE in Sengunthar College of Engineering and Technology, Anna University in 2005 and M.E (Computer Science Engineering) in A.S.L.Pauls College of Engineering and Technology, in the year 2013. He is currently working as Assistant Professor in Department of Computer Science Engineering in RVS Educational Trust's Group of Institutions. His area of specialization is Soft Computing and published papers in several National and International Conferences and Journals