International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

Implementation of Watch Dog Timer for Fault Tolerant Computing on Cluster Server

Meenakshi Bheevgade, and Rajendra M. Patrikar

***Abstract*—**In today's new technology era, cluster has become a necessity for the modern computing and data applications since many applications take more time (even days or months) for computation. Although after parallelization, computation speeds up, still time required for much application can be more. Thus, reliability of the cluster becomes very important issue and implementation of fault tolerant mechanism becomes essential. The difficulty in designing a fault tolerant cluster system increases with the difficulties of various failures. The most imperative obsession is that the algorithm, which avoids a simple failure in a system, must tolerate the more severe failures. In this paper, we implemented the theory of watchdog timer in a parallel environment, to take care of failures. Implementation of simple algorithm in our project helps us to take care of different types of failures; consequently, we found that the reliability of this cluster improves.

***Keywords*—**Cluster, Fault tolerant, Grid, Grid Computing System, Meta-computing.

I. Introduction

IN cluster environment during parallel application execution if any of the node failed, the computation cannot be completed because task allocated to that node remains incomplete. This is mainly because the other nodes are not aware of this failure and thus do not take care of such task. It is necessary that a computation should be continued despite the failure of individual node for reliable execution of parallel programs. A technique is required for detecting and responding to node failures by allocating job to the different node so that any application should be executed properly. Such technique is called fault tolerance technique [1],[2], and [3], which many times results in lower performance because of the overheads required for re-allocation of job. These systems are subject to graceful degradation mode. However, it is important to know the reliability of the system. A gracefully degradable system is one in which the user does not see errors except, perhaps, as a reduced level of system functionality. Current practice in building reliable systems is not sufficient to efficiently build graceful degradation [10] into any system. In a system with an automatic reconfiguration mechanism, graceful degradation becomes fairly easy to accomplish. After each error is detected, a new system reconfiguration is done to obtain maximal functionality using remaining system resources, resulting in a system that still functions, albeit with lower overall utility. Thus, Graceful degradation may define as a property that enables a system to continue operating properly in the event of the failure of some of its nodes. If its operating quality decreases at all, the decrease (D) is proportional to the severity of the failure (SF) (i.e. $D \propto SF$).

Although fault tolerant clusters are being researched for some time now, implementation of the fault tolerance architecture is a challenge. There are various types of failures which may occur in the cluster. Prediction of failure mechanism [3],[4],and [5] is very difficult task and strategies based on a particular failure mode may not help.

Meenakshi B. Bheevgade is with the Visvesvaraya National Institute of Technology, Nagpur, Maharashtra State - 440010, India (phone: 091-0712-2801291, e-mail: mbbhivgade@vnit.ac.in).

Rajendra M. Patrikar, PhD, was with Visvesvaraya National Institute of Technology and is now with CRL, Pune, Maharashtra State, India (e-mail: rajendra@computer.org).

II. Different Types of Failures

There are Different types of failures that are encountered in the system.

1) Hardware Failures – It need physical attention and human intervention for replacing the unit / element or part of the system.

2) Software Failures – Failures occur in operating system, or in process or at the time of application computation.

The two basic definitions of a failure have been stated. The first can be stated as the termination of the ability of the system as whole to perform its required function. The second one has stated that the termination of the ability of any individual component or process to perform its required function but not the termination of the ability of the system as a whole to perform.

In order to complete the computation properly availability of the system is also an important issue. A system fault can be caused by internal or external factors. Examples of internal factors could include specification and design errors, manufacturing defects, component defects and component wear out. Regardless of how well a system is designed, or how reliable the components are, failure cannot be eliminated completely. However, it is possible to minimize the impact on a system.

An error is the occurrence of a system fault in the form of an incorrect binary output. If an error prevents a system (or process) from performing its intended functions, a failure has taken place. If the incorrect data causes the system to crash or reboot then the error becomes a failure.

The system outages are classified into two categories.

1) Unplanned System Outages (failures): Unplanned outages are the result of uncontrollable random system failures, which occurs within hardware component (or process)..Unplanned system outages can be minimized through cluster environment.

2) Planned system outages (maintenance)-Planned outages should be scheduled to have a minimum availability impact on a system. Planned outages are the result of maintenance events revolving around repair, backup or upgrades system. Repairs are intended to remove faulty components and restore a system to a functional state.

III. General Strategies for Handling Faults

In general, it is difficult for user to monitor a faulty node. Typically, fault detection [4] can be done by some sort of response and respond method. Assuming that the node has been faulty and not responding, in such cases the software testing has done to re-check that the node has been responding. If the fault is transient then rechecks are to be done and if repeated re-checks show the failure of the system then the system is re-booted. If the fault persists after re-booting of the system then the node implied to be faulty and has to be removed from the list of available nodes. For this, the two algorithms have been studied.

Fault Detection Algorithm

A fault detection algorithm has been conveniently classified according to the time of the application with respect to normal operation of the system.

- Initial testing is required prior to normal use and serves to identify system elements containing imperfections, which introduces during the programming application.
- Concurrent detection, takes place simultaneously with the normal operation of the system
- Pre-emptive detection takes place after normal operation has been temporarily interrupted.
- Redundancy testing serves to verify the various forms of protective redundancy and takes place either concurrently or at scheduled intervals.

Recovery Algorithm

When fault is detected, the recovery algorithm has invoked to recover the system. We classified this recovery technique into four classes:

- Recovery of original performance
- Recovery of degraded performance
- Execution of safe shutdown
- Recovery by fault masking

The fault tolerance design evaluation [6] and [7] has performed by means of simulation, experiments or combination of all these techniques. The reliability prediction of the system has compared to that of the system without fault tolerance. Physical parameters, quality of fault detection and recovery [1],[2],[3],and [5] algorithms has used as parameters in generating reliability predictions. When degradation of performance takes place during recovery, reliability predictions need to be generated for various levels of performance. A different evaluation is needed when the reliability includes a specification of a minimum number of faults that are to be tolerated, regardless where in the system occurs.

In addition to the above algorithm, we also studied the theory of watchdog timer. The theory is as follows-

Watch Dog Timer

A watchdog timer[8],[9] is a computer hardware timing device that triggers a system reset, if the main program does not respond due to some fault condition such as a hang or neglects to regularly service the watchdogtimer. The intention is to bring the system back from the hung state into normal operation. Watchdog timer may be more complex, attempting to save debug information onto a persistent medium; i.e. information useful for debugging the problem that caused the fault. The watchdog timer does not report completion of its information saving task within a certain amount of time, the system will reset with or without the information saved. The most common use of watchdog timer is in embedded systems, where the specialized timer is often a built-in unit of a microcontroller.

Watchdog timer may also trigger control systems to move into a safety state, such as turning off motors, high-voltage electrical outputs, and other potentially dangerous subsystems until the fault is cleared.

The watchdog timer has been utilized for a fault tolerant technique in many serial applications. Usual implementation of this technique requires hardware implementation of timer/counter that usually interrupts CPU for corrective actions.

IV. Implementation of the Code

We have implemented the fault tolerant technique (we called this technique as watchdog timer algorithm) for a cluster by writing routines on a master server. The method implemented in our project includes re-checks to take care of transient faults included in the initial allocation phase. The cluster environment had booted by using only available nodes. If the failure had detected while allocating the task, then task had reallocated again. After allocation, the state of entire parallel application had checked at specific intervals of time-period as it had done by hardware watchdog timer. In this phase, master server node monitored an output data of the program at client node at specific time intervals. At the same time, the data had stored on the reliable storage on the local disk as well as on the master disk of the node. The storage had done for all the nodes in the available list of the nodes. If any of the node had not reported or not responding when a master node send a message to the all cluster environment nodes,

then that node had to be checked three times. In this phase, the master server reallocates the task if failure had detected. This was again to avoid the transient faults after certain interval of time. Then the task had allocated once again after the application had terminated. Here, the checkpoint has done thrice because of the following reasons:

1. If at first call, the CPU utilization of client node had 100%, it will not be able to respond about its existence. In second call, the client node may respond and update the status.
2. If the application crashed and it takes a lot of time for computation and still not responding or not able to save the data after specific interval of time, then the application gets terminated and resumes the application once again from the last saved point.
3. If link failure occurs due to network failure or due to some other reason and the node link had resumed once again then the client node, may respond to the query of the Master node and thus resume the application computation. Therefore, the next call may help the client to respond and update the status.

After the allocation phase starts, the client nodes had checked at regular intervals. If the allocation of job failed or the computation of the application failed due to network failure or node failure or some hardware component failure, then the flag had set to 1, indicates that the node will not be usable for the computation. Lastly, within stipulated time-period, if any of the above options was not feasible and there was no response from client node, the Master node will drop the current node from the list, add a new node and resume the job from the last best check-pointed state on this new node. It was possible to calculate the checksum that indicates correct behavior of a computation node. The watchdog timer algorithm will periodically test the computation with that checksum.

The algorithm has given in following pseudo code:

1. Boot the Cluster environment by using available nodes.
2. Execute the application in Cluster Environment.
3. Check at regular intervals of time.
4. Drop the node, if not responding.
5. Add a new node and allocate the job to the newly added node.
6. Calculate the total time required to execute the program.

The server node collects the data, which it determine the "state" of the client node when queried by the server.

V. Results

Watch dog timer algorithm was implemented in a SMP cluster based on the Linux. The LAM MPI system is implemented. The Monte Carlo application has tested in a parallel mode. The time required for the execution without applying the watchdog timer algorithm and no failure occurs, it takes about 768.10 seconds for execution. The watchdog timer algorithm was implemented and no failure occurs, the time taken by the application for computation is 793.94 seconds. If the two values had compared, then we had seen that the time required was not so high. After the implementation of watchdog timer algorithm and one failure had injected into one of the node, the application had still completed the computation but the time required was 940.175 seconds.

Th graph shown in Fig. 1 indicates the time required for completing the computation when with and without WatchDog Timer algorithm is implemented.

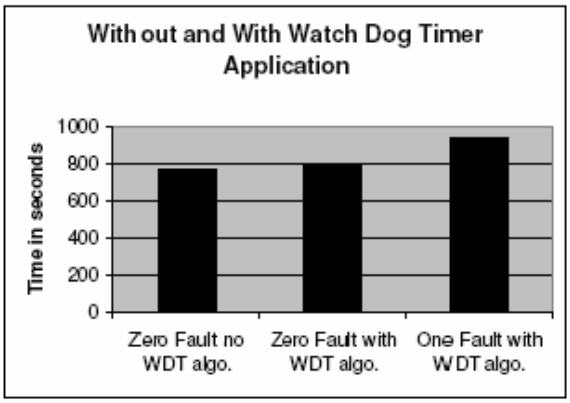

Fig. 1The graph between the computation and the faults encountered when the fault tolerance mechanism is applied

VI. Conclusion

In this paper we have implemented a fault detection and avoidance in a cluster based on the watchdog timer. This method can take care of hardware as well as software faults. The actual implementation of this method helped to detect the faults. The application had resumed on the newly added node from the last saved data. Thus helps to recover the computation with the help of watchdog timer algorithm. Thus using the fault detection algorithm, recovery algorithm and watchdog theory we are able to improve the new algorithm. Thus, this method helps us to improve the reliability of the application although the performance may degrade slightly because of computation overheads.

Acknowledgment

M. Bheevgade thanks Dr. C. S. Moghe for helpful discussions. She also thanks Ms. M. Ghoshal for correcting the English Grammer in this paper.

References

[1] Ian Foster and A. Iamnitchi,"A problem –Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems", IEEE, p.4-13 2000.
[2] Ian Foster, C. Kesselman, Craig Lee, G.v.Lazzewski,,"A Fault Detection Service for Wide Area Distributed Computations", Cluster Computing, v.2 n.2,p.117-128, 1999.
[3] Sriram Rao, Lorenzo Alvisi, Harrick M.Vin , "Egida : An Extensible Toolkit For Low-overhead Fault-Tolerance, Fault-Tolerant Computing", Digest of Papers. Twenty-Ninth Annual International Symposium, p. 45-55, 1999.

[4] Paul Toenend and Jie Xu, ”Replication-based Fault-Tolerance in a Grid Environment”, citeceer, 2003.
[5] Pascal Felber, Proya Narasimhan, Member, IEEE, “Experiences, Strategies, and Challenges in Building Fault-Tolerant CORBA Systems”, IEEE transactions on Computers , Vol.53, NO.5, May 2004.
[6] Object Management Group, “Fault Tolerant CORBA (Final Adopted Specification)” CMG Technical Committee Document formal/01-12-29.,Dec., 2001.
[7] R.Friedman and E.Hadad, “FTS: A High Performance CORBA Fault Tolerance Service”, Proc. IEEE Workshop Object Oriented Real-time Dependable Systems., Jan. 2002.
[8] Jack G. Ganssle, “Great Watchdogs”, V-1.2, Gaanssel Group, updated January, 2004.
[9] http://en.wikipedia.org/wiki/Watchdog_timer
[10] http://en.wikipedia.org/wiki/graceful degradation