ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

A Fault Tolerant Token Based Atomic Broadcast Algorithm Relying On Responsive Property

Mr. Neelamani Samal[1], Mr. Debasis Gountia[2], Mrs. Madhusmita Sahu[3]

Assistant Professor, Dept. of Computer Science & Engineering, Gandhi Institute For Education And Technology, Bhubaneshwar, India[1]

Assistant Professor, Dept. of Computer Science & Application, College of Engineering and Technology, Bhubaneshwar, India [2]

Lecturer, Dept. of Computer Science & Engineering, Gouri Sankar Residential College of Education, Bhubaneshwar, India[3]

Abstract: In the Distributed Environment where shared resources are involved, we have basically two types of mechanism to allocate the shared resources: either by passing tokens or by having Request and Reply Messages. In the shared environment, a processor might fail (i.e. may crash which may lead to failure). This paper proposes a fault tolerant token based atomic broadcast algorithm which does rely on unreliable failure detectors. It combines the failure detector and a token based mechanism, satisfying responsiveness property. The mechanism can tolerate processor level faults as compared to the existing system level failure, because the proposed system is relying on the unreliable failure detector and also rely on the responsive property.

Keywords: Mutex, Responsive Property, Critical Section, Token, Fault

I. INTRODUCTION

The requirement for highly reliable and available services has been continuously increasing in many domains for the last decade. Several approaches for designing fault tolerant services exist. The focus of this chapter is on software replication. Replication allows a number of replicas to crash without affecting the availability of the service**.** Systems involving multiple processes are often most easily programmed using critical regions. When a process has to read or update certain shared data structures, it first enters a critical region to achieve mutual exclusion and ensure that no other process will use the shared data structures at the same time. Mutual exclusion (often abbreviated as Mutex) algorithms are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections (CS). A critical section is a piece of code in which a process or thread accesses a common resource continuously. A program, process, or thread can have the critical section in it without any mechanism or algorithm which implements mutual exclusion.

Atomic broadcast / Total order broadcast ensures that messages broadcast by different processes are delivered to all destination processes in the same order in which they are initiated. Atomic Broadcast enables ordering mechanism and tolerates failure. In asynchronous system with crash failures two most widely used mechanisms to tolerate failure in Atomic Broadcast algorithm are 1) Unreliable Failure Detector 2) Group Membership Function.

An Unreliable Failure Detector does not provide consistent information about the failure status of processes. A Group membership service provides consistent membership information to all the members of a group. For example tell a process P that R has crashed, and to process Q that R is alive, at the same time. The overhead of a wrong failure suspicion is high when using a group membership. With group membership service, a wrong suspicion can lead to two costly membership operations: 1) removal of a process followed by, 2) addition of another process.

With a failure detector, neither the removal nor the addition of process is needed. Atomic broadcast algorithms based on a failure detector can be used to implement the group membership service.

II. CONTRIBUTION OF THIS PAPER

This paper is a contribution to the ongoing research on the different fault tolerant mutual exclusion algorithms. The research done so far is done on the system level, where a number of systems are interconnected to a grid network. The existing system needs a number of computers and a huge backbone of network connection to establish the

ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

experimental setup. And the experimental set up may suffer from any kind of unwanted failures that may be the failure of node or the communication network in either of the case we have to focus more on the communication network then on the node failure.

Here in this paper we focus on the process level failure in the system, which is a prototype of the real life communication system. In this paper, the existing features of the multi-core processor to simulate the behaviour of the grid computing environment by considering a small processor group are used.

This paper is an extension and modification of the Richard Ekwall [1] atomic broadcast algorithm, which was based on the unreliable failure detector. Here the focus is on the atomic broadcast of the process which will be using the unreliable failure detector and also satisfying the responsive property (RP). The quality of service [3] is also maintained in the course of failure detection process.

The failure is considered to be of crash [4] type. We are preparing a fault tolerant system which will definitely respond within a specified time limit, otherwise the checking for the failure or fault in the distributed system will be stopped and the process that does not reacted or obtained the resource will be declared as faulty and further that process will be removed from the Quorum. In this paper the main focus is on the tolerance level of the fault as well as on the atomic broadcast of the token which must be responded by other processes in a specified time interval.

III. PROPOSED MODEL

The goal of designing the proposed scheme is to simulate the fault tolerance capability of a distributed system within a well specified time interval.

The major concern of this approach is to have a

Fault tolerant system: The system should be very sure and confirm that the system is either faulty or not faulty. It should confirm the status of the processor to either faulty or not faulty. If the processor is not crashed and able to perform its tasks properly then the processor is not faulty, otherwise faulty.

Token Based: The systems that take part in the communication process should be able to enter into the critical section for execution only if it is having a valid token. So the processor that is having a valid token should enter into the critical section and other processors should try to acquire the token if it wants to enter into the CS.

Responsive property: The whole system should response to the user query and check for the workability of a process within a specified time limit. The individual processor should try to execute the CS within the time span specified i.e. the processor are getting a time limit within which the individual process should response. If any processor is not responding for the 1^{st} time it should try to respond 2^{nd} time and so on till it reaches the time limit specified.

IV. TERMINOLOGIES

We use the following terminologies to describe the fault tolerance in the distributed system.

The client is the individual processes that is responsible for performing the inter process communication. The client will be communicating to the local scheduler or the job manager for carrying out the operation. Here the client is the individual entity which is responsible for executing a specified task. The client or the processor should try to access the distributed task for execution. Here we assume that the client or the processor can only fail by crash. No other means of failure is accepted.

The Scheduler or the job manager prepares the ready queue, which indicates which client will get access to the shared resource in the network. The scheduler takes the responsibility for providing each process a fair chance to execute the task by accessing the distributed resource.

The worker is associated with individual scheduler or with the job manager. Each job manager has its own set of workers for performing the task.

ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

Each scheduler has its own set of workers but may be associated with any number of clients. The distributed scheduler or the job manager take the responsibility of scheduling the user queries and it can do so by distributing its whole work among the workers. The communication procedure of the client and the worker through the scheduler is shown in the following figure 1.

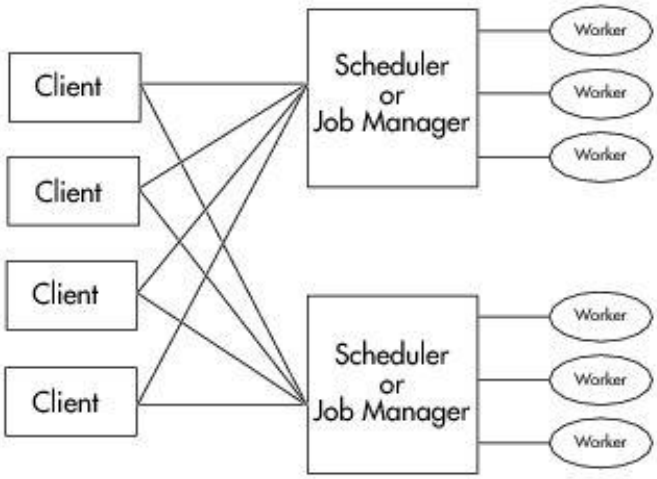

Figure 1. Configuration with Multiple Clients and Job Managers

V. PROPOSED ALGORITHM

To implement the fault tolerant atomic broadcast algorithm we need to follow the following steps

1. Divide the Job into a number of Tasks(LABS)
2. The task holding token Access the Resource
3. Start the timer for monitoring execution time
4. If the task responded within timeOut

   Declare the Process as OK

   else

   Declare the Process as Faulty

   & Remove the process/task from ready Queue

VI. IMPLEMENTATION DETAILS

To implement the experimental setup in a single computer to simulate the behaviour of the distributed computing process we need the following configuration of the machine, which will facilitate us to see the behaviour of multi processor environment.

The multi core processor enables to perform different segments of a single work inside a single processor. The segments can be created and executed by a process called segmentation. The individual segments have the capability to distribute the whole job into small segments which can proceed independently .If there will be any inter dependency then the local scheduler will be handling the issues of task dependency and any issues of deadlock or starvation.

Each individual task should have access to the primary memory and they should be executing their portion of work quickly. To facilitate this process we need at least 2GB of RAM. The more primary memory we have the faster

ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

the task can be executed. The execution time or the running time of individual task can be enhanced by increasing the capacity of the primary memory.

We need a configurable scheduler to allocate the resource to each task. The scheduler will be holding the responsibility of allocating the resource to each distributed task of the individual job. For the purpose of allocating resource to each task we can have a dedicated scheduler like High Performance Computing (HPC) Scheduler or we can also use the existing Scheduler present in the MATLAB. To implement and test the current working condition of fault tolerance mechanism, we have used the existing scheduler present in the MATLAB, the Load Sharing Facility (LSF) scheduler. The LSF scheduler is a pre-emptive scheduler. It allocates and manages the resources as per their priority. Here we are simulating the entire working environment in the MATLAB environment with proper configuration that is needed to create virtually a multiprocessor environment and study the behaviour of the whole system.

VII. SIMULATION & RESULTS

To simulate the experimental setup, we need to first create and configure a matalabpool which will enable us to simulate the multiprocessing environment, where we can have a distributed shared resource that each and every processor can access. So for that, we first create a local scheduler and define the maximum size of the cluster for the distributed task execution. We also need to set the path of the distributed job. This can be done by following the configuration option given below.

Here after setting the cluster size, path of the shared resource, and the path of the default scheduler present in the MATLAB we need to validate the configuration setting, to check the workability of the whole system as a unity. The configuration should pass through all the steps for its correct working condition. For an i5 machine we can have at best a cluster size of 8.

Initially the entire core of the machine has been divided into a number of partitions. Here each partition is a client, which is having the capability of performing the distributed task independently. The clients will be trying to access the shared resource to execute its CS within a specified time span. Each client is an independent task which executes the CS. In each run, the processor divides the whole task into a number of sub groups. For the case of a magic matrix, we divide the whole 8 x 8 matrix into 4 sub parts of 4 x 4 matrices. So that each task can proceed faster than, the original 8 x 8 matrix operation. This kind of divisions helps to make better and faster utilization of the existing resource in an effective manner.

From the observation, it's very clear that the execution time of the above calculation by the individual processor or the client in the system as a whole is less than the whole task. The execution time for dividing the whole task and the actual calculation can be compared for performance measurement. The effect of distributed task scheduling and the performance enhancement by the division of the whole task into a number of sub groups, we find that the sub groups can run and execute faster than the system if it is taken as a whole.

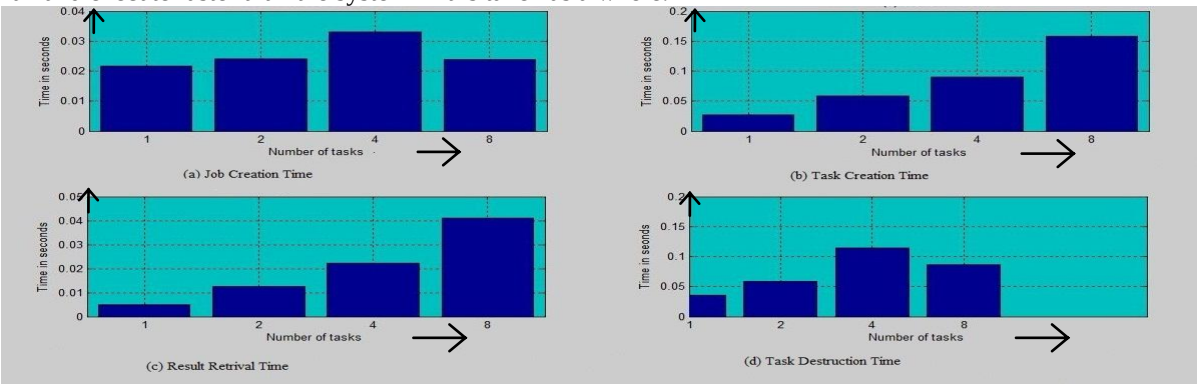

Figure 2. Simulation Result

We have simulated all the results using MATLAB (7.10.0.499 (R2010a)) in Intel® Core (TM) i5 CPU M460 @2.53GHz machine.

According to this work, we have created twenty jobs and assigned them to a local scheduler. Once the task has been submitted to the local scheduler, the scheduler will be deciding which job will be executed. The job needs to be

ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

selected from the pool of twenty jobs. The job that is selected will be performing its execution. The time taken by the client to create a job with varying number of tasks is shown in the figure 2(a). Here the plot shows the total amount of time taken by an individual processor for task creation. As we can see from the plot the time taken to create a job with increasing number of tasks is reducing with respect to the increasing number of task. So the job creation time will reduce substantially with the increasing number of tasks of a specified job.

The simulation result 2(b) shows the task or worker creation time in the local scheduler. According to this graph the time taken to create a distributed partition with increasing number of workers is reducing. This indicates that by dividing the entire core into more sub groups the time for creation of task is gradually reducing and the capacity of handling the tasks concurrently is increasing. So by increasing the cluster size of the machine the processor can run faster than in normal condition. By following the above plot we can confirm that the distributed tasks can execute faster than the normal task.

After the division of the whole job into a number of tasks we allow them to execute the CS (Fig 2(c)). The local scheduler will first randomly allocate the token to a task. The task that is having the token will get a fair chance to execute the CS. The processor or the task that have token should access the shared resource before the timeout is called. For this particular case I have allocated 20 milliseconds time for the CS execution. If any task is not responding for the 1st attempt in spite of having the token, then it should not stop there, by declaring the processor is faulty rather it will try till the timeout event is reached. If the task is still not able to access the shared resource, with having the token, then that process should be declared to be faulty. Here we assume that the crashed processor will only be not responding in spite of having a valid token. And once the processor completes its execution it should release the token to other processor that want to enter into the CS.

After the task has completed its execution it should be destroyed from the memory, enabling other processors to access the distributed shared resource. The figure 2(d) shows the time taken by the controlling site to destroy the jobs form the local scheduler.

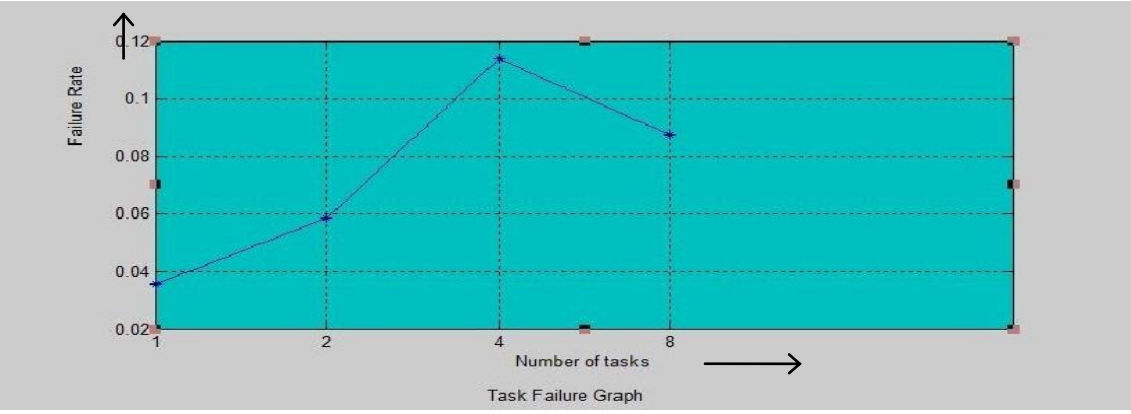

Figure 3. Performance Measurement

From the figure 3, we can see that the task failure rate was initially increasing with smaller number of tasks in a job. As the number of tasks increases the failure rate is gradually declining. Because at the initial stage, it may wrongly suspect a process to be faulty, but as per our proposed model the processor should not be declared to be faulty unless the processor is not at all responding within a specified time period. Here we are eliminating wrong suspicion of fault of a processor by allowing the task to execute its CS by acquiring the token, by sending continuously request for token to all other sites in the request queue. As the wrong fault suspicion is reducing and only the actual faults are identified the proposed model is having better performance that the normal fault tolerant mechanisms.

VIII. CONCLUSION AND FUTURE SCOPE

In this paper we have developed a process level fault tolerant mechanism which is based on the unreliable failure detector and the atomic broadcast mechanism which should be able to detect the failure within a specified time period to make an improved fault tolerant mechanism. It combines the failure detector and a token based mechanism, satisfying responsiveness property. The mechanism can tolerate processor level faults as compared to the existing

ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

system of system level failure, because the proposed system is relying on the unreliable failure detector and also rely on the responsive property.

There still remain many open problems in the search for efficient and accuracy in the fault tolerance mutual exclusion algorithms.

These are some of the interesting open problems like:

- Implementing Deadlock Prevention while Resource Allocation rather than using the Deadlock avoidance which is not so efficient for distributed system which is increasing the overhead.
- At present we are having a single control site for resource allocation, but we can have a Distributed Controlling Site for Resource Allocation for making the fault tolerance mechanism livelier.
- Introduction of Intelligence system for fault Detection so that the time taken for detecting a fault can be minimized and optimized.

REFERENCES

[1] Ekwall, R., Schiper, A., "*A Fault-Tolerant Token-Based Atomic Broadcast Algorithm*", IEEE transactions on dependable and secure computing, VOL. 8, NO. 5, September October 2011.

[2] Token Passing Bus Access Method, ANSI/IEEE Standard 802.4, 1985.

[3] Chen, W., Toueg, S., and Aguilera, M. K., "*On the Quality of Service of Failure Detectors*", IEEE Trans. Computers, vol. 51, no. 2 pp. 561-580, May 2002.

[4] Defago, X., Schiper, A., and Urban, P., "*Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey*", ACM Computing Surveys, vol. 36, no. 2, pp. 372-421, Dec. 2004.

[5] Ekwall, R., "*Atomic Broadcast: A Fault-Tolerant Token Based Algorithm and Performance Evaluations*", Ph.D thesis, Ecole Polytechnique Federale de Lausanne, May 2007.

[6] Ekwall, R. and Schiper, A., "*Comparing Atomic Broadcast Algorithms in High Latency Networks*", Technical Report LSR-Report-2006-003, Ecole Polytechnique Fe´de´rale de Lausanne,Switzerland, July 2006.

[7] Friedman, R. and Renesse, R. V., "*Packing Messages as a Tool for Boosting the Performance of Total Ordering Protocols*", Proc. Sixth IEEE Symp. High Performance Distributed Computing, pp. 233-242, Aug. 1997.

[8] Schiper, A. and Toueg, S., "*From Set Membership to Group Membership: A Separation of Concerns*", IEEE Trans. Dependable and Secure Computing, vol. 3, no. 1, pp. 2-12, Jan.-Mar. 2006.

[9] Urban, P., Shnayderman, I., and Schiper, A., "*Comparison of Failure Detectors and Group Membership: Performance Study of two Atomic Broadcast Algorithms*", Proc. Intl Conf. Dependable Systems and Networks (DSN), pp. 645-654, June 2003.

[10] Chang, J.M. and Maxemchuck, N., "*Reliable Broadcast Proto-cols*", ACM Trans. Computer Systems, vol. 2, no. 3, pp. 251-273, Aug. 1984.

[11] Maxemchuk, N.F. and Shur, D.H., "*An Internet Multicast System for the Stock Market*", ACM Trans. Computer Systems, vol. 19, no. 3, pp. 384-412, Aug. 2001.

[12] Amir, Y., Moser, L., Melliar-Smith P., Aggarwal, D., and Ciarfella, P., "*The Totem Single-Ring Ordering and Membership Protocol*", ACM Trans. Computer Systems, vol. 13, no. 4, pp. 311-342, Nov.1995.

[13] Whetten, B., Montgomery, T., and Kaplan, S. M., "*A High Performance Totally Ordered Multicast Protocol*", Proc. Dagstuhl Seminar on Distributed Systems, pp. 33-57, 1994.

[14] Cristian, F., Mishra, S., and Alvarez, G., "*High-Performance Asynchronous Atomic Broadcast*", Distributed System Eng. J., vol. 4, no. 2, pp. 109-128, June 1997.

[15] Cristian, F., "*Reaching Agreement on Processor Group Member-ship in Synchronous Distributed Systems*", Distributed Computing, vol. 4, no. 4, pp. 175-187, Apr. 1991.

[16] Larrea, M., Arevalo, S., and Fernandez, A., "*Efficient Algorithms to Implement Unreliable Failure Detectors in Partially Synchronous Systems*", Proc. Intl Symp. Distributed Computing, pp.34-48, 1999.

[17] Hadzilacos, V. and Toueg, S., *"Fault-Tolerant Broadcasts and Related Problems"*, Technical Report 94-1425, Dept. of Computer Science, Cornell Univ., May 1994.

ISSN (Print) : 2320 – 9798
ISSN (Online) : 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 2, April 2013

[18] Chandra, T. D. and Toueg, S., "*Unreliable Failure Detectors for Reliable Distributed Systems*", J. ACM, vol. 43, no. 2, pp. 225-267, 1996.

[19] Chockler, G. V., Keidar, I., and Vitenberg, R., "*Group Communication Specifications: A Comprehensive Study*", ACM Computing Surveys, vol. 4, no. 33, pp. 1-43, Dec. 2001.

[20] Shye, A., Moseley, T., Reddi, V. J., Blomstedt, J.," *Using Process-Level Redundancy to Exploit Multiple Cores for Transient Fault Tolerance* ".

BIOGRAPHY

Neelamani Samal received the Bachelor of Computer Science and Engineering degree from Jagannath Institute for Technology & Management, Parlakhemundi, India. He received the Master of Technology degree in Information Technology from the College of Engineering & Technology, Bhubaneshwar, India. Since February 2010, he has been a Faculty with the Gandhi Institute for Education and Technology, Bhubaneshwar, India. He is having around 03 years of teaching and research experience. His research interests include Operating System, Data structures, Software Engineering and Distributed Systems.

Debasis Gountia received the Bachelor of Computer Science and Engineering degree from University College of Engineering, Burla, India. He received the Master of Technology degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur, India. Since January 2006, he has been a Faculty with the College of Engineering & Technology, Bhubaneshwar, India. He is having around 10 years of teaching and research experience. His research interests include cryptography, data structures, high performance computing architecture, formal language and automata theory, operating system, and distributed systems.

Madhusmita Sahu received the Master in Computer Application from Utkal University, Bhubaneshwar, India. Since February 2010, she has been a Faculty with the Gouri Sankar Residential College of Education Bhubaneswar, India. She is having around 05 years of teaching and research experience. Her research interests include Operating System, Data structures, Software Engineering, Data Mining and distributed systems.