Volume 2 No. 3 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

A Multiple Fault Tolerant Approach with Improved Performance in Cluster Computing

Sanjay Bansal[1], Sanjeev Sharma[2], Rajiv Gandhi Prodhyogiki Vishwavidya [3]
[1]Medi-Caps Institute of Technology and Management, Indore India
[2] School of Information Technology, [3] Bhopal, India
[1] sanju2.bansal@gmail.com, [2] sanjeev@rgtu.net

ABSTRACT

In case of multiple node failures performance is very low as compare to single node failure. Failures of nodes in cluster computing can be tolerated by multiple fault tolerant computing. In this paper, we propose a multiple fault tolerant technique with improved failure detection and performance. Failure detection is done by improved adaptive heartbeats based algorithm to improve the degree of confidence and accuracy. Failure recovery is based on reassignment of load with a rank based algorithm Performance is achieved by distributing the load among all available nodes with dynamic rank based balancing algorithm. Dynamic ranking algorithm is low overhead algorithm for reassignment of tasks uniformly among all available nodes. Message logging is used to recover message loss.

Keywords: *Message Passing Interface (MPI), Distributed Scheduling, Interprocess Communication (IPC), Multiple Faults, Failure Detection, Failure Recovery.*

1. INTRODUCTION

Distributed Computing uses multiple geographically distant computers and solves computationally intensive task efficiently **[GG, 08].** There are certain strong reasons that justify using distributed computing in comparison to single powerful computer like mainframes. Cluster computing is a one way to perform distributed computing. Several computing nodes connected together form a cluster. Several Loosely coupled clusters of workstations are connected together by high speed networks for parallel and distributed applications. Cluster computing offers better price to performance ratio than mainframes. If one machine crashes, the system as a whole can still survive in distributed system. Computing power can be added in small increments in distributed systems. In this way incremental growth can be achieved. Cluster computing has increased in popularity due to greater cost-effectiveness and performance. Recent advances in processors and interconnect technologies have made clusters more reliable, scalable, and affordable **[RB, 99]**.

Fault-tolerance is an important and critical issue in cluster computing. Due to very large size of distributed system and complexity of computation the chances of fault are more. As the size of computational clusters increases, mean time to failure decreases drastically. In such a situation, inclusion of fault tolerance is very essential. There are mainly three types of approaches of fault tolerance in cluster computing. Hardware based fault tolerance is very costly. Software algorithm is only possible when source codes are available. Software layer based overcomes all these problems. In a software layer based fault tolerance, mechanism works as a layer between application and system. This fault tolerance scheme is independent of the cluster scalability as well as fully transparent to the user **[TS, 08]**.

Several approaches are proposed by researchers. L. Kale´ and S. Krishnan proposed Charm++ **[LK, 93].** CHARM++ is a portable concurrent object oriented system based on C++. Zheng et al. discuss a minimal replication strategy within Charm++ to save each checkpoint to two “buddy” processors **[GZ, 04]**. Chakravorty et al. add fault tolerance via task migration to the Adaptive MPI system **[SC, 06] [SC, 07] [CH, 06]**.Yuan Tang et al. proposed checkpointing and rollback **[YT, 06]**. However, this system’s main drawback is expensive in terms of time and overhead. Their system relies on processor virtualization to achieve migration transparency. John Paul Walters et al. proposed replication-Based fault tolerance for MPI application **[JPW, 09]**. However issues related to replication like consistency among replica and encoding overhead are need to be address carefully. Our approach is based on fault tolerance using adaptive and dynamic failure detection as well as improved degree of confidence and accuracy. Recovery is based on reassignment of tasks to all available nodes using a rank based algorithm. It reassign the task uniformly among the entire node hence performance is improved. Rank based algorithm is low overhead, require minimal communication among nodes and efficient.

2. CLUSTER COMPUTING

A cluster is a type of parallel or distributed processing system which consists of a collection of interconnected computers cooperatively working together as a single, integrated computing resource **[RB, 99]**. Cluster is a multi-computer architecture. Message passing interface (MPI) or pure virtual machine (PVM) are used to facilitate inter process communication in a cluster.

3. PURPOSED SYSTEM FAULT TOLERANT ARCHITECTURE FOR MULTIPLE NODE FAILURE IN CLUSTER

There are many types of failures that can occur in a complex distributed system. For example, a node can fail, a network connection could fail, or a disk could fail. Furthermore, these failures may occur one at a time, or several may happen at once. In this paper, we focus on detecting individual node failure and network failure in so far as they appear to be node failure.

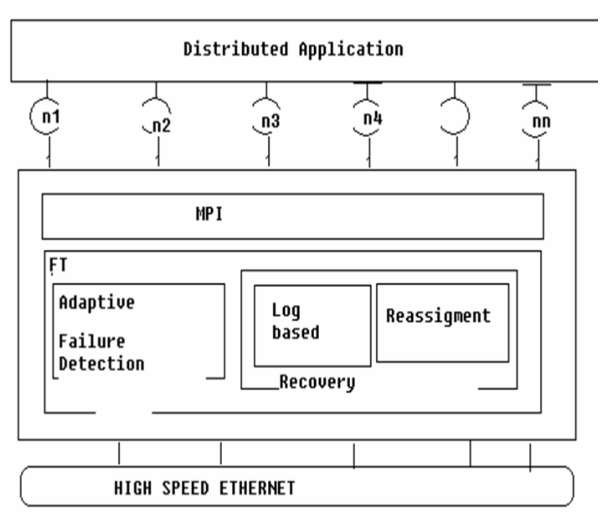

Fig -1 Architecture of proposed Technique

Multiple nodes failure fault tolerance with performance is achieved by two modules. Failure detection and recovery based on reassignment of load.

3.1 Failure detection

Fault detection algorithm is based on the heartbeat strategy. In this strategy every failure computing node periodically sends a heartbeat message to a coordinator node or sibling node to inform them that it is still alive. When a coordinator node fails to receive a heartbeat from another node for a predetermined amount of time (timeout) it concludes the remote node failure. The most desirable of a failure detector is that it should not detect a working node as a failure and it should not trust a failure node.

Setting a fix time out of a simple heart beat can affect the accuracy. It can declare a healthy node as a crashed due to shorter timeout. While all other or some nodes can respond in specified fix time out because of low traffic and load, some healthy nodes may not response due to heavy traffic. In such situation fix time out failure detector will detect a failure node even it is not failed one **[TD, 96]**. In order to solve the problem Researcher suggested adaptive failure detector **[CF, 01]**. W. Chen *et. al* proposed a different approach based on a probabilistic analysis of network traffic **[WC, 02]**.However this uses a constant safety margin which requires an optimization. Hayashibara et al. proposed a method using an assumption that the inter-arrival time followed a normal distribution **[NH, 04]**. A different approach of failure detector is accrual failure detector. Accrual failure detectors consists of detector modules that associate, to each of the monitored processes, a real number value that changes over time. The φ-failure detector samples the arrival time of heartbeats and maintains a sliding window of the most recent samples. The window is used to estimate the arrival time of the next heartbeat **[XD, 03].** However, the proposed failure detectors are poorly adapted to very conservative failure detection because of their vulnerability to message losses. In practice message losses tend to be strongly correlated (i.e., losses tend to occur in bursts). A proposed accrual detector designed to handle this problem is the k-failure detector **[TK, 04].** The k-failure detector takes into account both messages losses and short-lived network partitions, each missed heartbeat contributing to raising the level of defined suspicion according to a predetermined scheme.However normal distribution does not work with large scale and unstable distributed system. Xiong et al. proposed exponential distribution failure detection **[NX, 08]**.Researchers proposed setting of adaptive time out based on probabilistic behavior **[NX, 09].** Our approach is different in the sense that time out is based on seven different load and traffic level. These are seven fuzzy situation are defined and corresponding time out are set accordingly. These seven fuzzy logics of load and traffic are minimum (min), normal, medium, average; moderate high, high and very high. Values of very high traffic and load can be easily set by analyzing the design specification supplied by the vendor. Alternatively these values can be calculated by analysis through load and traffic engineering [YJS, 09]. Time out is maximum for very high load and traffic condition. If a node do not respond after maximum time out corresponding very high load and traffic logic then node is detected as a failure node.

Algorithm 1: Failure detection for node failure

```
# define  traffic_load_rank tlr_ low, tlr_min, tlr_normal,
tlr_medium, tlr_average, tlr_moderate_high, tlr_very_high
traffic_load_rank
# define timeout  t_min, t_normal, t_med, t_avg, t_mh,
t_very_high;
For each node  Pi,   0≤i≤ n-1
{
set traffic_load_rank=min_val;
set timeout as per traffic_load_rank
sends message “I am ok!”;
while (traffi_load_rank is less than very high)
{
 If (timeout expires for Pi and)
{
if (no_ answer) then
```

Volume 2 No. 3 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

```
{
increase traffic_load_rank = traffic_load_rank +1;
time out= next higher_time_out;
continue;
}
}
put Pi in suspect list;

if (answer)
{
 remove Pi from suspect list;
store traffic_load_rank of pi in i^th element of
traffic_load_rank _array;
traffic_load_rank =min_ traffic_load_rank;
brake;
}}
if (no_naswer and traffic_load_rank =max_
traffic_load_rank)
{
declare Pi as a failure node;
store failure node i in i^th element of failure_node_array
}}
```

3.2 Recovery

Recovery is done with in two module; recovery of message loss and reassignments of load to remaining system.

3.2.1 Recovery of messages loss:

To recover message loss we are using low overhead uncoordinated checkpoint, distributed message logging. This fulfills the requirement to tolerate n concurrent faults (n is number of concurrent faults).

Pessimistic message logging is used for message loss due to multiple node failure. This technique is already examined by MPICH-V2, a public domain platform implementing pessimistic logging with uncoordinated checkpoint **[MA, 06]**.

3.2.2 Reassignment of task to remaining system:

Load of all working nodes is obtained as an out put of failure detection algorithm as rank. Load of all failure nodes is obtained by message logging technique used in this paper. After failure detection of nodes and recovery of message loss, available loads of all failure and working nodes is distributed uniformly among all available nodes. This reduces the overall execution time of system. For this we will use rank based algorithm **[SB, 10]**. Proposed algorithm is based on assigning a rank based on their load. This algorithm is based on following assumptions and condition for distributed environment:

- Jobs are independent
- All jobs are same nature as per communication and computation.
- All computing nodes having huge and equal computing capability.

3.2.2.1 Informal Description of the reassignment algorithm:

In this paper distributed scheduling with a rank based approach is purposed having following major components:

- **Basic load cum rank table generator module.**
- **Load distributor module**
- **Job generator**

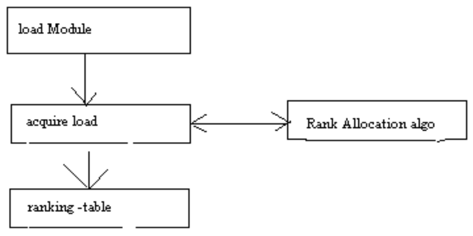

FIG 2: Basic Architecture of Rank Allocation Algorithm to different Processor with different Loads

- **Basic load cum rank table generator module**:

Load information module collects the load of different computing nodes. Based on load information**,** processors are ranked. Ranking is based on giving the lowest rank to a processor having least loaded node and highest to the node having highest load. In this way, a rank is allocated to each node based on their load value. A higher rank is allocated when node is heavily loaded. A lower rank means node is lightly loaded. A load is transferred by load transfer module between a highest rank and lowest rank**,** a second highest ranked node to second lightly ranked node and so on.

This module uses Rank allocation algorithm shown in Algorithm 2 to generate the Rank table of figure 3. This rank based algorithm assign rank to computing node based on their load values. A heavy loaded node will be assigned as a higher rank and a lightly loaded will be assigned to a lower rank.

Algorithm2: Rank allocation:

```
While (node available) do
    -   sort all node in ascending order of their load
        values
    -   allocate rank 1 to first node and 2 to second and
        so on
```

- store the value of least rank and highest rank in two variables least_ rank and highest_rank

End

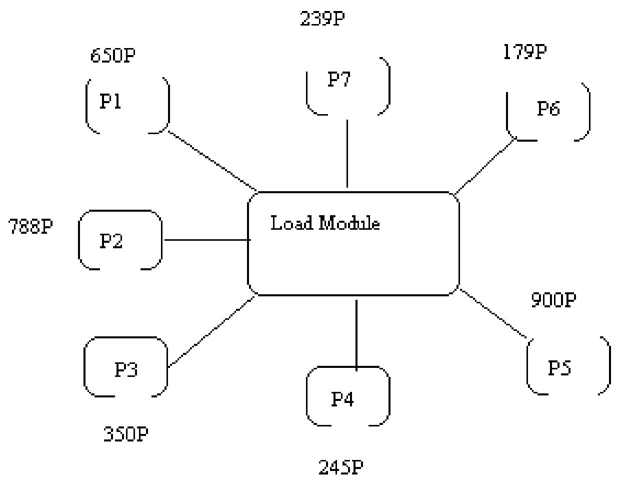

Fig3: Load module connected to different nodes working in distributed computing system

By using the Rank algorithm 1, a rank table is generated. For figure-2, load table is generated as shown in Table 1

Table 1: Rank Table generated for figure 2 by rank algorithm 1

CPU-ID	LOAD	RANK
P1	650	5
P2	788	6
P3	350	4
P4	245	3
P5	900	7
P6	137	1
P7	239	2

- **Load Distribution Module**

Once Rank load is prepared by the load module, job scheduler will transfer the process from first rank to last rank, second to second last rank node and so on. This will do by Load transfer algorithm2.

Algorithm 2: Load Distribution Algorithm

```
Load_distraibution ()

{

var last=get_highest_rank ();

For (rank=1; rank=last; rank++)

{

avg_load=load [rank] +load [last]

Load_to_transfer=load [last]-Avg_load

Transfer load from last rank load to current node by
Load_to_transfer;

Last=last-1;

Rank [last] =Rank [rank];

}

Allocate rank () ;}
```

Algorithm 3: Load Transfer algorithm

- **New job cum failure node jobs assignment module**

First search the least ranked node and transfer job to it using following algorithm

```
Allocate_processor_new_job ()
{
While (job available ())
{
    Node=get_leats_rank_node ();
    Compute (Node, job);
}
```

Algorithm 4: Algorithm for new job arrival

4. EXPERIMENTAL RESULT AND PERFORMANCE

4.1 Results of Adaptive Failure detection

We have designed a simulator in java for our adaptive failure detection. Results obtained by the simulator are in following table.

Table 2: Results comparison obtained for failure detection

S.N	No. of Nodes	Number of False nodes detection	
		Simple heart Beat Algo.	Adaptive Heart Beat Algorithm
1	4	1	0
2	8	3	1
3	16	5	2
4	32	7	4

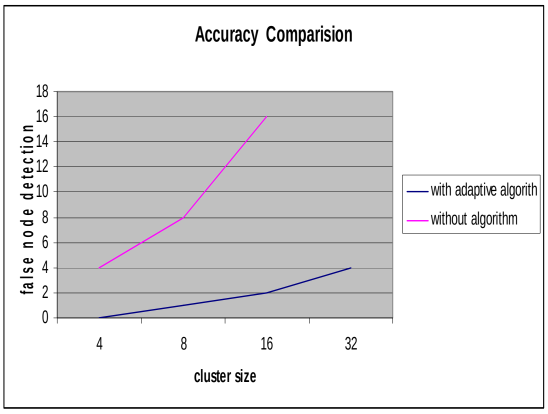

Fig 4: Results comparison obtained for failure detection

4.2 Experimental Results obtain after recovery

We have simulated our result on computer nodes of 4 computers. All computers had equal computing power. We had written a process using MPI for printing hello. An experimental result shows reduction in response time due to uniform distribution of tasks of all working and non working node by proposed recovery scheme.

Table 3: Performance comparison after recovery with uniform load distribution

Load Allocation To All 4 Nodes	Response Time in (ms)	
	Purposed Method	Simple Mpi
5,8,30,43	630	275
3,15,47,55	682	299
3,16,55,72	742	356
0,44,78,88	992	405

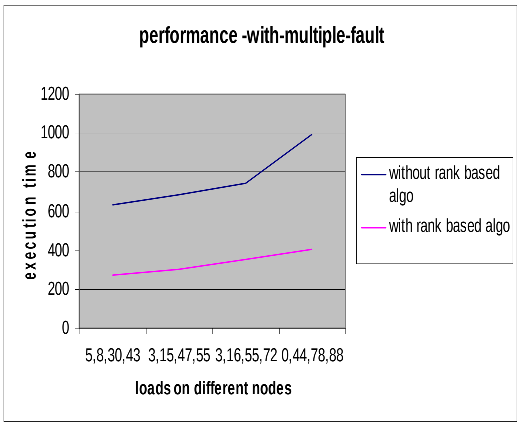

Fig 5: Performance comparison after recovery with uniform load distribution

5. Conclusion

Multiple faults with performance depend upon accurate detection and fast recovery with performance. In this adaptive heartbeat strategy is used. This algorithm will remove problem of degree of confidence. Accuracy is improved. Recovery is based on reassignment of tasks. Reassignment is based on distributing the load uniformly among all working node. It reduces the response time. Rank based algorithm uses high splitting ratio, hence convergence is very fast. It takes less number of iteration and time. Rank based algorithm which is simple and effective. Algorithm proposed having less execution time, fast convergence and fewer messages overhead as compared to other algorithm purposed. Although this algorithm is suitable for homogenous environment but it can be further extended to heterogeneous environment.

REFERENCES

[GG08] G. Georgina., "D5.1 Summary of parallelization and control approaches and their exemplary application for selected algorithms or applications," LarKC/2008/D5.1 /v0.3, pp 1-30.

[RB,99] R.Buyya. "High Performance Cluster Computing: Architectures and Systems," Vol. 1, Prentice Hall, Upper Saddle River, N.J., USA, 1999.

[TS, 08] T.Shwe and W.Aye, "A Fault Tolerant Approach in Cluster Computing System," 8-1-4244-2101-5/08/$25.00 ©2008 IEEE.

[YT, 06]Y.Tang, G. Fagg and J. Dongarra, "Proposal of MPI operation level checkpoint/rollback and one implementation," Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 0-7695-2585-7/06 $20.00 © 2006 IEEE.

[LK, 93] L. Kale´ and S. Krishnan, "CHARM++: A Portable Concurrent Object Oriented System Based on C++," Proc. Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA' 93), pp. 91-108, 1993.

[CH, 06] C. Huang, G. Zheng, S. Kumar, and L.V. Kale´, "Performance Evaluation of Adaptive MPI," Proc. 11th ACM SIGPLAN Symp.Principles and Practice of Parallel Programming (PPoPP '06), pp. 306-322, 2006.

[SC,06] S. Chakravorty, C. Mendes, and L.V. Kale´, "Proactive Fault Tolerance in MPI Applications via Task Migration," Proc. 13th Int'l Conf. High Performance Computing (HiPC '06), pp. 485-496, 2006.

[SC, 07] S. Chakravorty and L.V. Kale´, "A Fault Tolerance Protocol with Fast Fault Recovery," Proc. 21st Ann. Int'l Parallel and Distributed Processing Symp. (IPDPS '07), pp. 117-126, 2007.

[GZ, 04] G. Zheng, L. Shi, and L.V. Kale´, "FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI," Proc. IEEE Int'l Conf. Cluster Computing (Cluster '04), pp. 93-103, 2004

[JPW, 09] John Paul Walters and Vipin Chaudhary, "Replication-Based ault Tolerance for MPI Applications," IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 20, NO. 7, JULY 2009.

[AB, 03] Aur´elien Bouteiller, Franck Cappello and Thomas H´erault, MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging" SC'03, November 15-21, 2003, Phoenix, Arizona, USA

[MA, 06] Mehdi Aminian, Mohammad k. Akbari, Bahman Javadi, "Coordinated Checkpoint from Message Payload in Pessimistic Sender-Based Message Logging, "1-4244-0054-6/06/$20.00 ©2006 IEEE.

[TD,96] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 43(2): 225-267, Mar. 1996.

[CF, 01]C. Fetzer, M. Raynal, F. Tronel, "An adaptive failure detection protocol", *8th IEEE Pacific Rim Symp. undependable Computing*, 2001, pp. 146 - 153

[WC, 02] W. Chen, S. Toueg, M.K. Aguilera, "On the quality of service failure detectors", IEEE Transactions on Computers, 51(2), 2002, pp. 13 - 32.

[NH, 04] N. Hayashibara, X. Defago, R. Yared, and T. Katayama. The phi accrual failure detector. In Proc. 23rd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS2004), pages 66-78, Florianpolis, Brazil, Oct. 2004.

[NX, 08] N. Xiong. Design and Analysis of Quality of Service on Distributed Fault-tolerant Communication Networks. Doctoral thesis, Japan Advanced Institute of Science and Technology, March, 2008

[NX, 09] N. Xiong, Athanasios V. Vasilakos, Laurence T. Yang, and Lingyang Song, Pan Yi, Rajgopal Kannan, Y. Li, "Comparative Analysis of Quality of Service and Memory Usage for Adaptive Failure Detectors in Healthcare Systems," IEEE Journal on Selected Areas in Communications (JSAC), vol. 27, no. 4, pp. 495-509, May 2009.

[AS, 09] Abderrahmane Sider and Raphaël Couturier, "Fast load balancing with the most to least loaded policy in dynamic networks," J Supercomput (2009) 49: 291–317,DOI 10.1007/s11227-008-0238-5.

[MAR,07] Md. Abdur Razzaque and Choong Seon Hong," Dynamic Load Balancing in Distributed System: An Efficient Approach,"

[SB,10] S. Bansal and S. Sharma "An Improved Multiple Faults Reassignment Based Recovery in Cluster Computing," Journal of Computing, vol. 2 November 2010.

[YJS,09] Y. Jayanta Singh, and Suresh C. Mehrotra, "An Analysis of Real-Time Distributed System under Different Priority Policies," World Academy of Science, Engineering and Technology 56 2009

[XD,03] X. Defago, N. Hayashibara, T. Katayama, "On the Design of a Failure Detection Service for Large-Scale Distributed Systems", *Intl. Symp. Towards Peta-bit Ultra Networks* (*PBit 2003*), 2003, pp. 88 - 95.

[TK,04] N. Hayashibara, X. Defago, T. Katayama, "Flexible Failure Detection with K-FD", Research Report IS-RR-2004-
006, 2004.