A Design and Implementation of Cluster Heartbeat Network for Efficient Fault Detection

Ahmad Shukri Mohd Noor, Emma Ahmad Sirajudin
School of Informatics and Applied Mathematics,
Universiti Malaysia Terengganu, Terengganu, Malaysia
ashukri@umt.edu.my

***Abstract*—To achieve fault tolerance in a server cluster, fault detection capability is a primary prerequisite. Efficient fault detection is prompt, correct and complete. This paper revisited the technique called Reactive Failure Detection (RFD) that dynamically predicts a heartbeat delay from a cluster node. We also identified the requirements to deploy RFD in actual servers. A new cluster heartbeat network with concurrency is proposed to use push and pull interaction during live monitoring and determining node's status. The prototype of the new model is tested on a platform running multiple independent web applications and analyzed for its implementation and design correctness.**

***Index Terms*—Heartbeat Network; Fault Detection; High Availability; Concurrency.**

I. Introduction

Failure detection is an important design consideration for providing high availability in a generally distributed system. This process involves isolation and declaration of a fault to enable proper recovery actions to start. It is a prerequisite to failure recovery in distributed system [1][4][5].

Many different techniques are used to detect failures, ranging by different efficiency and complexity [1]. Correct, prompt and efficient failure detection is the requirement to a recovery mechanism that is able to do self-recovery discreetly and without external party intervention. As a result, a fault tolerant service is realized.

Often, there is tradeoff from weaknesses of either fast detection with low accuracy or completeness in detecting failures but with a lengthy timeout [3][4]. For instance, the failures can be detected quickly but the probabilities of false faults are high. On the other hand, the failures can be detected completely but after a long time resulting in delay of recovery.

The approach to failure detection in a distributed system called Reactive Failure Detection (RFD) was introduced [4]. In RFD, heartbeat interaction is used to monitor the health of servers and an expectation of heartbeat arrival is maintained to detect a failure within an adaptive timeframe and subsequently confirming it using ping. RFD finds an optimal value, H_max to dynamically predict the heartbeat delay by considering the changing environments to ensure the fault is promptly detected and at the same time to avoid over-detection.

In a cluster, each node's live heartbeats are used to draw the behavior of the current network and CPU usage. When a new heartbeat is inconsistent with the node's expected behavior found with RFD, a fault may have occurred and will be checked before the suspicion is confirmed. The requirements to implement RFD are concurrency programming and a heartbeat network within a cluster of nodes. The nodes in the cluster are closely monitored from the periodic heartbeat messages that they send to a monitoring service node, namely Heartbeat Monitor (HBM). When a particular node fails to send a heartbeat message within the estimated time, HBM will suspect a failure. It then reconfirms the failure by pinging the node.

Section 2 revisits the RFD algorithm and discusses its requirements. Section 3 describes the proposed design of the cluster heartbeat network. Section 4 describes its implementation and lastly section 5 present results for discussion.

II. Reactive Failure Detection (RFD)

In [4] an adaptive technique for failure detection was introduced. This technique incorporates pinging to ensure the liveliness of a node once it is suspected for failure thus is affirmative. This technique performs a central sampling on the heartbeat inter-arrival time to obtain the estimation for the next heartbeat arrival. If the next heartbeat did not arrive within this timeframe, the monitor raises a state of suspicion and sends a ping echo request to the monitored node. The threshold for the heartbeat to arrive reflects the current state of the node CPU load and network condition. The RFD technique is given by the formula:

$$H_{max} = \sum_{i=1}^{n} S_i + S_{n+1}$$

where: H_{max} is the maximum heartbeat arrival time
$\sum_{i=1}^{n} S_i$ is the total time elapsed (total heartbeat time before the last heartbeat)

and

$$S_{n+1} = \frac{\frac{\sum_{i=1}^{n-1} S_i}{|S_{n-1}|} + S_n}{2}$$

where: $\sum_{i=1}^{n-1} S_i$ is the total heartbeat time in S_{n-1}
$|S_{n-1}|$ is the size of the sampling in S_{n-1}
S_n is the inter-arrival of the most recent heartbeat arrived

Considering again, p is monitoring q and is waiting for the next heartbeat $(n+1)$ from q. The probability of the $q(n+1)^{th}$ heartbeat is influenced by the last heartbeats. From the analysis of heartbeat inter-arrival time, the last heartbeat S_n has a significant likelihood to resemble the next heartbeat S_{n+1}therefore is factored by 50% for the next heartbeat while the rest of in sample S is factored by 50%. This can ensure a close reflection of the current condition of the monitored node and network. A deviation can be detected based on this reflection and will be confirmed by pinging to make sure it is a permanent failure instead of a temporary glitch that occasionally happens due to network or CPU load.

The Reactive Fault Detection (RFD) component gives timely detection of node failure with completeness and high accuracy. The RFD is designed to be dynamic by deploying an estimated time of arrival (ETA) threshold that adapts to the network and server condition. It can deliver higher availability by having an intuitive fault measure that can avoid false detection and enables a timely recovery. A false detection can trigger unnecessary recovery and put dispensable load on network and server which will result in waste of resources.

Heart beats sent over the network sometimes are affected by network bandwidth and load. Therefore, it is necessary to consider network delay. In the beginning the server initialization will take some delay that will gradually reduce with some minor irregularities. Over time the prediction value will closely assimilate the server and network current states. Any changes of the states can be detected promptly based on the prediction.

The fundamental requirement for the RFD implementation is high concurrency and separated tasks that can communicate with each other as well as a heartbeat network within the cluster for live monitoring of nodes. In this paper, the design and implementation taken is by using structured programming and interrupt signal libraries.

III. PROPOSED DESIGN

Cluster Heartbeat Network

The purpose of a heartbeat network is to enable real time communication between a monitor and the nodes. Heartbeats are sent via dedicated sockets for each node. A node indicates its aliveness by sending periodic heartbeats to the monitor. With the RFD technique, the monitor is proactive where it performs a central sampling to estimate the incoming heartbeat. When the estimation has elapsed, the monitor raises a suspicion of the node failure if no heartbeat is received. Therefore, one of the concurrent processes needed is to find the optimal H_{max} to predict the next heartbeat. Another concurrent process is the timer that would count down the delay provided by H_{max} so that a fault can be detected immediately within the timeframe.

The flowchart in Figure 1 describes the proposed flow of fault detection program. The flowchart describes the process in the while loop. First of all, in the loop the flag TIMEUP is polled to see if it is set indicating a timeup has happened in previous loop. If it is, the ping request is sent to the monitored node to confirm its status. If ping request returns node unreachable the failure is confirmed. At this point the program will enter recovery mode. If otherwise, ping reply is received the node is confirmed to be still alive and the Heartbeat Monitor (HBM) program will clear the TIMEUP flag indicating it is no longer a suspicion. The threshold value is reset to initial value to begin resampling. If the loop is entered and TIMEUP flag is clear it means that previously the node was acting like expected i.e. no time up did happen.

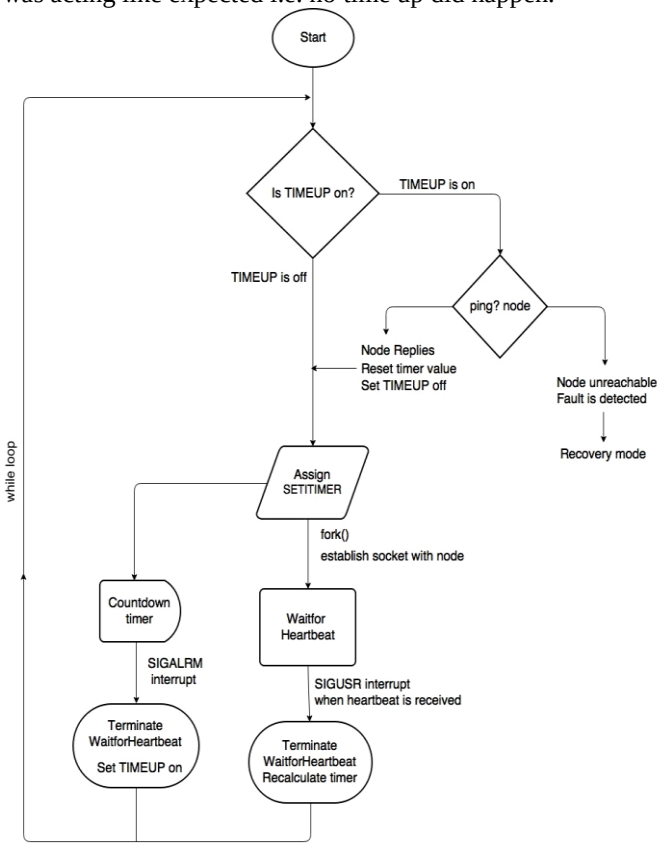

Figure 1: Flowchart of Fault Detection Process

In this case the program goes straight to assigning the timer with the threshold value. The threshold value was calculated in previous receive signal handler if it is not the initial value. Subsequently, a new child process is created. Inside the child process, socket connection is reestablished. After that it will listen indefinitely to the socket for heartbeat message. After a message is received it then checks if it is the right heartbeat message. If it is indeed the right message, the child process sends a signal to parent process to make interrupt to program. The signal handler is entered and in here the time is stamped to obtain the heartbeat inter-arrival time. Also the threshold value is recalculated. If the heartbeat did not arrive in time, the timer will elapse and this will cause interrupt to program. The program will enter the handler and in here the TIMEUP flag is set. The process continues for each loop.

A. Process Duplication

High concurrency is needed for both processes because RFD is time sensitive and precision is important. Concurrent computation brings complexity in implementation and different techniques give different level of concurrency. Perceived concurrency is when tasks seem to be running simultaneously but in fact they are not as they take turn to run, saving one process's state to only continue after another process completes. This is the case when multithreading is done on single-core processor. In this paper however, the approach is process duplication instead of multithreading. Therefore, it achieves true concurrency on single core machine. In process duplication, concurrency is achieved by deploying new processes from the original process. Interaction among the processes is enabled using interrupt signals. The important considerations are identified as following:

i. In RFD implementation, interaction between processes is required because a process will need to stop the other when certain events have occurred while both are run concurrently. That is, when a heartbeat is received, the timer should be unset whereas if the timer elapses before any heartbeat is received, the monitoring activity should be stopped.
ii. Also in RFD, the processes need to use the same resource clock for their complementary computations. In the one process, time taken for the heartbeat to come is calculated, while the second process will signal if the time taken in the first process is exceeding H_{max}. If the processes are run on different cores as in parallel programming, the clock rates might be slightly different. For this reason, the time computation must be done at one process or core only to achieve precision.
iii. Race condition is a common problem in concurrency programming. Processes or threads that use same memory may change it while others are still using it. Precautions must be taken to avoid this as it can give wrong results in RFD calculation. Using process duplication, this is avoided naturally because after a process is duplicated, it has its own copies of variables inherited from original process.

B. Process Termination

Process termination is necessary in the proposed design in two situations. Firstly, when a heartbeat has arrived, the timer process should be canceled and secondly when the timer has elapsed, the process should stop waiting for heartbeat as a suspicion for fault needs to be serviced. Practically, two parallel processes will cancel the other when one of two events occurs first.

C. Concurrent Tasks

The algorithm in Figure 2 describes the fault detection process. There are two tasks that must be run concurrently. By sending a signal, the task that gets to finish first will terminate the other task and determines the mode in the next loop; whether to continue monitoring or begin suspecting the node. In this implementation, the tasks are developed in C language using a number of POSIX libraries.

The program loops for the continuous monitoring of heartbeats. In each loop two concurrent processes are started; the waiting of heartbeat messages, and the timing of the waiting process.

```
//Start while loop:
initialize H
create socket
if (TIME UP flag is set) ; ping node
if (node echo reply) ; status = OK
else status = FAIL; initiate recovery
else if (TIME UP flag is clear)
set H to alarm timer
create child process; accept socket connection with node
just wait for heartbeat
if (correct heartbeat message arrived)
send receive signal to parent process
else notify that wrong message is received
//End while loop

//Signal Handler 1: For heartbeat receive:
unset timer to stop alarm
update sampling and the next expected value for H
terminate child process

//Signal Handler 2: For timer elapsed for heartbeat expectation:
unset timer to stop alarm
set TIME UP flag
terminate child process
```

Figure 2: Algorithm for Heartbeat Monitor

Parent process will call *fork()* to create a new child process in each loop. In the new child process socket connection is reestablished and timer restarted for the heartbeat expectation lapsing in the parent process. Subsequently, the child process is terminated by the parent either because heartbeat has been received or not been received within time. If the heartbeat arrives in time, a signal called SIGUSR1 is sent to the parent process which will be serviced by Signal Handler 1. Otherwise, if the timer elapsed before any heartbeat is received, a signal called SIGALARM is generated by the timer class to the parent process which will be handled by Signal Handler 2. In Signal Handler 1 and Signal Handler 2, parent process generate terminating signal called SIGTERM to the waiting process (child process).

D. Confirming Failure

Under some circumstances the node fails to send a heartbeat or a heartbeat simply cannot reach the monitor in time even when the node is running like usual. This could be due to CPU loads or network latency. In order to be precise and not draw a false presumption about the node, the monitor program will utilize ping command to determine the status of the missing heartbeat node. If a reply is received, the node is no longer in suspicion and the monitor program will reset the threshold to its initial value. It is necessary to reset the threshold and restart the monitoring process to draw a new assimilation of the network and server state as previous assimilation has been interrupted and is no longer relevant for the new state. On the other hand, if there is no reply and the ping utility concludes that the host is unreachable, the monitor program will declare the state of failure for the node and will enter a recovery mode.

IV. EXPERIMENT

In this experiment, the proposed cluster heartbeat network is tested in a distributed environment created with virtual machines on the hypervisor VMWare on a single-core, 8GB RAM machine. Using the hypervisor, a cluster of servers hosting web applications are connected on a private network to a monitoring server (for fault detection) and an indexing server (for fault recovery). Users are able to access the web applications through a proxy server that is also connected on the private network. For this purpose, several network types are specified on the hypervisor; which are NAT for the proxy and *Host-only* for the private network. The web servers also contain replication of each other in a setup called neighbor replication for the purpose of recovery. Ideally, when a node fails, its replica is activated somewhere else inside the cluster. On user side, these changes are masked as it happens behind the proxy.

Once live monitoring is started, the nodes begin sending heartbeats to the monitor. Initially, the values of maximum heartbeat delay, H_{max} are preset. It gradually changes to become closer to the actual heartbeat inter-arrival time. This is depicted in figure 4. It can be said, over time H_{max} gives representation of the network and CPU condition of the node. In effect, the increase in CPU load will cause more delay in heartbeat delivery. Ping latency is also affected by network condition and CPU load, however the prototype does not consider the latency in the fault detection calculation.

In this work, the failure detection is designed to respond to three failure causes. They are server total fault in which case the server is completely failed, network cut or instability which could be temporary or permanent or heartbeat generator malfunction. The monitor detects failure if socket accept returns fail for three consecutive times without having to confirm on ping echo reply. This is because server is still alive but not able to send heartbeat that could be due to port malfunction or heartbeat generator program hang/terminated. This is also a state of malfunction since no heartbeat essentially means monitoring cannot be performed. But it may not be necessary to invoke a fail-over recovery because server may still be alive. The failure causes were simulated to observe the results. In the first fault test, the node was stopped by pausing the VMWare player. In the second test, the network card on monitored node was shut down using terminal command line. In the last test, the heartbeat generator program was terminated during execution. All these fault simulations are detected promptly by the HBM program.

The recovery action is initiated after the failure has been detected. As a result, the service is restored from a different server and users do not experience significant downtime as the detection and recovery happen very quickly. It is observed that fault tolerance has been achieved

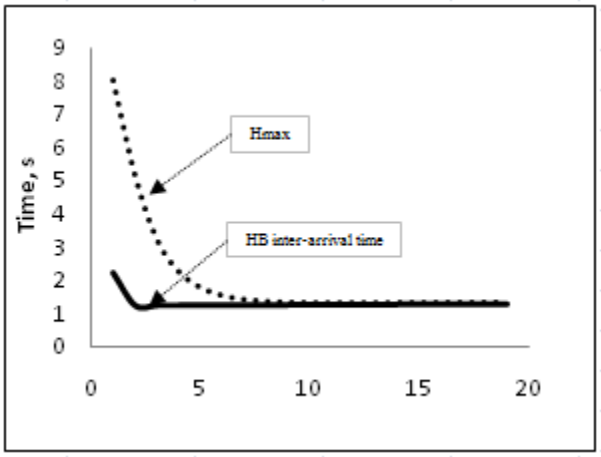

Figure 3: Expectation of next heartbeat adapting to inter-arrival time

V. CONCLUSION

Fault detection is the primary prerequisite to achieve a fault tolerant system. The efficiency of recovery also relies on the accuracy and timeliness of the fault detection. Efficient fault detection is prompt, correct and complete. The technique called Reactive Failure Detection (RFD) dynamically predicts a heartbeat delay from a cluster node. As a result, it is effective in changing environments. To deploy RFD, a cluster heartbeat network with concurrency is required. In this work, push and pull interaction is used during live monitoring and determining node's status. The prototype of the new model has been tested in a platform running multiple independent web applications and observed for its implementation and design correctness. Furthermore, with a recovery plan, a node failure is promptly recovered, giving uninterrupted service to users. The system that employed RFD technique with a recovery plan has been observed to become tolerant to node failures. The design and implementation of cluster heartbeat network to detect failures using efficient technique have been presented in this paper.

ACKNOWLEDGMENT

The research was supported by the Ministry of Science, Technology and Innovation (MOSTI) of Malaysia. (Grant No. 52074)

REFERENCES

[1] Falai, L. and Bondavalli, A. (2005), "Experimental Evaluation of the QoS of Failure Detectors on Wide Area Network," International Conference on Dependable Systems and Networks (DSN'05), pp. 624–633.
[2] Fu, S. (2010), "Failure-Aware Resource Management for High-Availability Computing Clusters with Distributed Virtual Machines," Journal of Parallel and Distributed Computing, vol. 70, pp. 384–393.
[3] Kaur, A. and Verma, S. (2015), "Performance Measurement and Analysis of High-Availability Clusters," SIGSOFT Softw. Eng. Notes, vol. 40, pp. 1–7.
[4] M. Noor, A. S. and M. Deris, M. (2012), "Fail-stop Failure Recovery in Neighbor Replica Environment," Procedia Computer Science, vol. 19, pp. 1040–1045.
[5] Noor, A.S.M., Deris, M.M. (2010), "Failure recovery mechanism in neighbor replica distribution architecture" Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and

Lecture Notes in Bioinformatics), 6377 LNCS (M4D), pp. 41-48. Springer Verlag
[6] Mamat, R., M. Deris, M., and Jalil, M. (2004), "Neighbor Replica Distribution Technique for Cluster Server Systems," Malaysian Journal of Computer Science, vol. 17, pp. 11–20.
[7] Mitchell, M., Oldham, J., Samuel, A. (2001), Advanced Linux Programming, pp. 45-60, 95-129, Indiana USA, New Riders Publisher.
[8] Schmidt, K. (2006), "High Availability and Disaster Recovery Concepts, Design, Implementation". Berlin London: Springer
[9] Shi, L., Yang, S. and Zhang, Q. (2010), "Research and Analysis of Adaptive Failure Detection Algorithm," 3rd International Symposium on Computer Science and Computational Technology, pp. 21–24, Academy Publisher.
[10] Zakaria, A., Awang, W., Mohamad, Z., Rose, A., and M. Deris, M. (2010), "Improving Response Time, Availability and Reliability Through Asynchronous Replication Technique in Cluster Architecture of Web Server Cluster," in Database Theory and Application, Communications in Computer and Information Science, vol. 118, pp 29-36, Springer
[11] Noor, A.S.M., Deris, M.M. (2009), "Extended heartbeat mechanism for fault detection service methodology" Communications in Computer and Information Science, 63, pp. 88-95. Springer Verlag
[12] Matsudaira, K. "Scalable Web Architecture and Distributed Systems" Architecture of Open Source Applications. http://www.aosabook.org/en/distsys.html. Accessed on 22 January 2015.
[13] Khan, F. G., Qureshi, K., and Nazir, B. (2010), "Performance Evaluation of Fault Tolerance Techniques in Grid Computing System," Computers & Electrical Engineering, vol. 36, pp. 1110–1122
[14] Butenhof, David R.(1997) "Programming with POSIX threads." Addison Wesley Professional.
[15] Lea, Douglas (2000) "Concurrent programming in Java: design principles and patterns". Addison-Wesley Professional