Survey on Scalable Failure Detectors

Shihui Song SHIHUI@STANFORD.EDU

Abstract

Maintaining a timely view of the current system status is essential to the performance and functionality of distributed systems. Failure detectors have long been essential to distributed systems. In this paper, we evaluate two failure detection algorithms specifically aimed at large-scale systems. Both assume fail-stop (non-Byzantine) models but the similarities end there. Dynamo's failure detector relies on pinging with a weak eventual completeness model based on randomization. On the other hand, the classic gossip-protocol is based instead on heartbeats with a strong completeness model. Our simulations test the conclusions of Dynamo's failure detector in order to evaluate its gains from the traditional gossip heartbeat style approaches. We end with remarks on the advantages of both algorithms and the systems that are best suited for each.

1. Introduction

The need for failure detectors arose early in the development of distributed systems to address the unreliability of asynchronous networks. It is aimed to aid consensus by maintaining group membership. Despite recent industry trends towards utilizing huge amounts of commodity hardware with commonplace failures, there is still the inherent need to respond and adjust to failures. Failure detectors must also adapt to the demands of large scalable systems with common failures.

1.1. Principles and Previous Work

Failure detectors in general make extremely few assumptions about the network. There is no bounds on the message delay and messages can be lost in transit. We do however, assume a fail-stop and non-Byzantine model. The clocks are each instance must also maintain a constant speed.

The two basic properties of failure detectors are spelled out below in the seminal paper by Chandra and Toueg (Chandra & Toueg, 1996):

Completeness There is a time after which every process that crashes is permanently suspected by {some/all} correct process. The completeness model is also divided into *weak* and *strong* consistency where a weak system only needs a single correct process to suspect failure but a strong system requires all correct processes to suspect the failure. *Crashes will be learned*.

Accuracy There is a time after which some correct process is never suspected by any correct process. *Crashes must be learned efficiently*.

Chandra and Toueg demonstrate the need for tradeoffs between completeness and accuracy in an asynchronous unreliable network. The first proposed failure detectors would choose completeness over accuracy. A model often helped by the heartbeat strategy.

There has been normally two mechanisms for detecting liveliness:

Heartbeat A node will send out messages every designated time period to alert others that it is still alive.

Pinging A node will ask other nodes whether they are alive, and if they reply in some timely manner, then the original node is satisfied that the other node is alive.

To increase the efficiency of using heartbeats, heartbeats will often be piggybacked or referenced from other network messages. However, the trend towards large distributed system have aroused the need for even more efficient failure detectors.

2. Failure Detectors

For the project, we introduce the two failure detectors. They are based on different detection principles and are from different eras as well.

2.1. Gossip Style

In one of the earliest failure detector papers, van Renesse introduces the gossip style failure detector (van Renesse et al., 1998). The model relies on a variation of heartbeats. Instead of broadcasting a heartbeat to every other node, a node will instead increase its own heartbeat and gossip its entire membership list to one random node during the time period. This reduces the message count from polynomial to linear. The receiver reconciles the membership lists by taking whichever version number is greater. If after a fail time period there's been no updated heartbeat number for a particular node, then that node is deemed failed.

2.1.1. Gossip Round Robin

Instead of the random node destination for sending membership list, the Gossip Round Robin (GRR) protocol chooses the target deterministically (Ranganathan et al., 2001). The naive protocol selects the destinations sequentially. The optimized protocol utilizes binary round-robin. As seen in the figure from the original paper. So the total time it takes for a node to have its membership be known throughout the entire system is only $log_2(N)$.

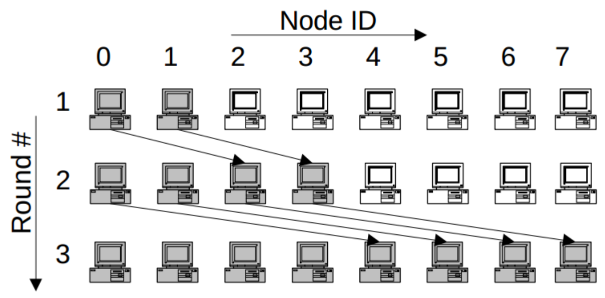


Figure 1. The Destination ID becomes Source ID + 2^{r-1} where r is the round number.

This also comes with the added benefit of deterministically knowing which nodes should be the sender for the supposed gossip message. Therefore, the protocol also allows for immediate failing of nodes should a gossip message not arrive.

2.2. Dynamo's failure detector

A scalable and efficient failure detector was introduced by Gupta and Chandra in 2001 (Gupta et al., 2001) but most notably adopted by Dynamo from Amazon (DeCandia et al., 2007), hence the terminology of calling it Dynamo's failure detector.

This failure detector relies on randomized pinging and distributed failure confirmation. A node is alive if it replies to the ping. If no ack has been received, the original node asks k other nodes to ping it as well . If none of the k has received a response, then that node is marked as failed by the original node.

2.2.1. Recovery

Recovery was never outlined in the original paper and does not appear in the models. However, for the presentation simulation, we decided on showing the recovery protocol that makes the most sense.

Heartbeat would be fast and acknowledged by all, but not efficient. It requires a broadcast message that is not used in either of the two protocols. On the other hand, pinging brings no guarantees of recovery. Pinging dead nodes to ask about recovery is also extremely inefficient especially with dead node counts rising. Hence, a single ping/heartbeat to all nodes alerts others upon recovery and thereby gains a list of alive nodes from ACKs will be the recovery protocol. Any nodes that hear of a ping from a deceased node will mark that node as alive again.

3. Implementation

The failure detectors are implemented in Java with individual JVMs representing each node. UDP is chosen for Dynamo's failure detector for ease of transmission where it's assumed that sending UDP packets on the localhost will result in negligible packet loss. However, due to the large messages for the gossip protocols, TCP has to be used. The heartbeat versions in the messages would be greater than the maximum size for a single UDP packet whereas TCP would re-assemble the packets with ease. A single period of 2 seconds was the unit of measurement for time to allowed for messages to be sent and processed at each round. Thread pools are utilized to maintain a reasonable computing consumption. Nevertheless, simulation of 1000 nodes for Dynamo's failure detector crashed my machine as well a corn machine. In the end, an ec2 instance of c4.8xlarge running Ubuntu was utilized.

4. Protocol Analysis

4.1. Failure Detection Delay

The Gossip protocol has a sigmoid detection function where the majority of the detection occurs after the failed time period (which was set to 10) for a size of

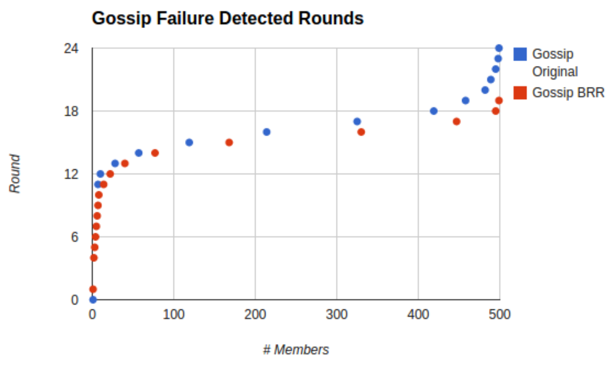


Figure 2. The failure detection time for the Gossip protocols with a system size of 500 and failure time period of 10 with a message loss of 0 for accuracy.

500. The quickness of the detection for most of the nodes is very impressive. On the other hand, the binary detection is only as good as the original in the most case. The receiver expecting gossip and failing immediately is only useful within the round that the original nodes is expected to make.

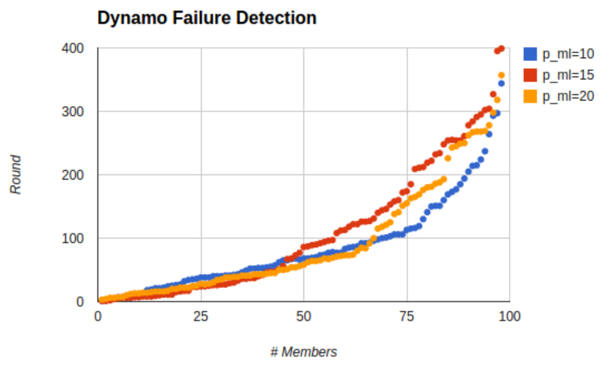


Figure 3. For different message loss rates, the graph shows the detection time needed for each member to detect the single loss.

Dynamo failure detector insists on performance based on *expected* failure delay and it's not difficult to see why. *Expected* delay is the time in which mathematically even with randomness, the failure should have been detected by other processes. The weak completeness requirement is easy satisfied despite randomness as proven in the paper and as we can see with the initial detections. However, strong completeness as shown in Figure 1 is exponential.

To compensate, Dynamo as a system will also receive updates from clients alerting them to failed nodes to replace or compensate for. Therefore, the absolute time for detecting failed nodes is not as important for Dynamo's failure detector.

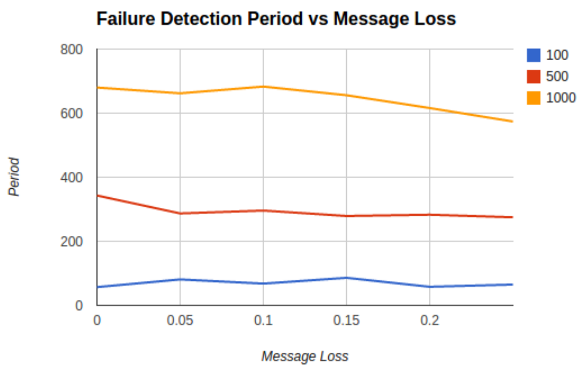


Figure 4. For different message loss rates, the graph shows the detection time needed for half of each system to detect the single loss.

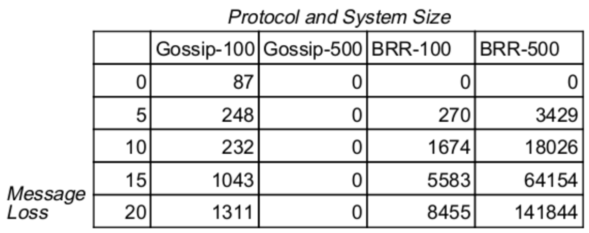
To its credit, Dynamo's failure detector is much more resilient towards message loss than the Gossip protocols are. As seen in Figure 4, the message loss percents do not contribute to any significant delays in failure detection time and instead even helps the system sizes of 1000.

One of the fundamental differences between the two protocol is the presumption of liveliness. The gossip protocols presume dead if not heard whereas Dynamo's failure detector presumes alive until pinged otherwise.

4.2. Mistaken Failures

The internet principle maintains that there's no certainty of why a node has not responded within a specified amount of time. There could have been a network partition, messages could been dropped, or the node could have failed. But the origin node has no way to guarantee what caused the unsuccessful return of message. Instead, failure detectors rely on timeouts to ascertain liveliness, which ultimately lead into incorrect detections.

Gossip protocols fluctuates according to message loss. The original protocol Is heavily dependent more on the fail period than the message loss. That is, even if messages are lost, the nodes could still wait for another one in time to update. Therefore, high fail periods such as the square root of the entire system size leads to NO mistakes for Gossip-500. However, because the fail period is so short for a size of 100, many mistakes were made. But even more sensitive is the Gossip Binary Round Robin protocol, and with good reason too. Any loss messages immediately denote a failed node. It's essential for a system implementing Binary Round Robin to have trivial message loss.

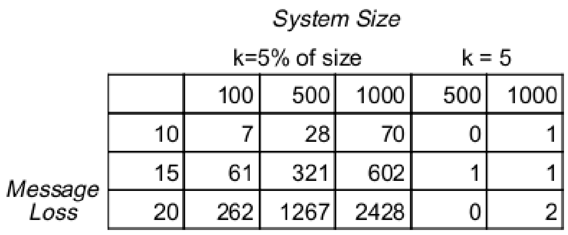


Message Loss	Gossip-100	Gossip-500	BRR-100	BRR-500
0	87	0	0	0
5	248	0	270	3429
10	232	0	1674	18026
15	1043	0	5583	64154
20	1311	0	8455	141844

Protocol and System Size

Table 1. Mistaken Failures with failPeriod set to $N^{1/2}$ for the original Gossip protocol and $log_2(N)$ for Binary Round Robin

Dynamo relies on k relaying hosts pinging the possible failed node to confirm failure. k is quite the precarious number and must scale with the number of nodes, but ever so delicately as the original paper proved. In Table 2, we see the number of mistaken failures for given system sizes and message loss probabilities. Notice that the total mistaken failures scale almost perfectly with the system size. Obviously, however, $k = 5$ is not ideal as the system sizes increase since mistakes also increase. We also see the unlikelihood of an incorrect detection in Table 2 where $k = 0.05N$ since the probability of no positive ACKs with more tries decreases sharply. Efficiency is traded off in this delicate situation.



	k=5% of size			k = 5	
Message Loss	100	500	1000	500	1000
10	7	28	70	0	1
15	61	321	602	1	1
20	262	1267	2428	0	2

System Size

Table 2. A table showing the total mistakenly failed nodes in the system after a 100 time periods based on different message loss, k, and system sizes.

4.3. Scalability

The Gossip has a simple network load pattern. Each round every working node sends out reconciliation message. The class we chose was of the following form:

$$\{\text{int fromPort}, \text{HashMap} < \text{Integer, Integer} > \text{ allCounters}\}$$

Therefore the total space it takes to send this as a string can be considered

$$4 \text{ bytes} + N * 2 * 4 \text{ bytes} = (8N + 4) \text{ bytes}$$

Therefore total size of messages sent for a round is:

$$N(8N + 4)\text{bytes}$$

There scuttlebutt variation of the gossip protocol that reduce the message size by only reconciling a limited number of the members, but it relies on a dual communication approach instead of a single push-gossip technique (van Renesse et al., 2008).

Dynamo's messages are much shorter with 5 simple integers (10 bytes):

$$\{\text{int type, int fromPort, int toPort, intperiod, int relayPort}\}$$

Even despite the message loss and subsequent k relay ports with the maximum 4 messages each, the average message per round given message loss percent p_{ml} is only:

$$N(2 + 4k(1 - (1 - p_{ml})^2)) \times 10\text{bytes}$$

Therefore, Dynamo's network usage is still linear per round whereas gossip's is polynomial. In fact, the 1000 system size is so large for message size that even the c4.8xlarge could not handle it.

5. Conclusion

Failure detectors maintain group membership but the gossip and Dynamo protocols are aimed at quite different systems. The gossip protocol has a short failure detection delay but extremely sensitive to message loss. The problem with network usage renders it only very useful to systems up to a certain size. On the other hand, the Dynamo failure detector gives no guarantee of failure detection delay but handled message loss solidly. Its scalability and unsureness makes it ideal for large systems that have a weak consistency model. As distributed systems continue to grow, failure detectors will also continue to flourish.

References

Chandra, Tushar Deepak and Toueg, Sam. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996. ISSN

0004-5411. doi: 10.1145/226643.226647. URL http://doi.acm.org/10.1145/226643.226647.

DeCandia, Giuseppe, Hastorun, Deniz, Jampani, Madan, Kakulapati, Gunavardhan, Lakshman, Avinash, Pilchin, Alex, Sivasubramanian, Swaminathan, Vosshall, Peter, and Vogels, Werner. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007. ISSN 0163-5980. doi: 10.1145/1323293.1294281. URL http://doi.acm.org/10.1145/1323293.1294281.

Gupta, Indranil, Chandra, Tushar D., and Goldszmidt, Germán S. On scalable and efficient distributed failure detectors. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*, PODC '01, pp. 170–179, New York, NY, USA, 2001. ACM. ISBN 1-58113-383-9. doi: 10.1145/383962.384010. URL http://doi.acm.org/10.1145/383962.384010.

Ranganathan, Sridharan, George, Alan D., Todd, Robert W., and Chidester, Matthew C. Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters. *Cluster Computing*, 4(3):197–209, July 2001. ISSN 1386-7857. doi: 10.1023/A:1011494323443. URL http://dx.doi.org/10.1023/A:1011494323443.

van Renesse, Robbert, Minsky, Yaron, and Hayden, Mark. A gossip-style failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, pp. 55–70, London, UK, UK, 1998. Springer-Verlag. ISBN 1-85233-088-0. URL http://dl.acm.org/citation.cfm?id=1659232.1659238.

van Renesse, Robbert, Dumitriu, Dan, Gough, Valient, and Thomas, Chris. Efficient reconciliation and flow control for anti-entropy protocols. In *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*, LADIS '08, pp. 6:1–6:7, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-296-2. doi: 10.1145/1529974.1529983. URL http://doi.acm.org/10.1145/1529974.1529983.