Failure Detection and Consensus in Large-Scale Cloud Systems with Gossip Protocols

Ritesh Narendra Ghorse
NC State University
Raleigh, NC
rghorse@ncsu.edu

Shreyas Muralidhara
NC State University
Raleigh, NC
schikkb@ncsu.edu

Tanvi Pandit
NC State University
Raleigh, NC
tmpandi2@ncsu.edu

Abstract

Fault tolerance is one of the key characteristics of distributed systems. Efficient failure detection reduces the recovery time and helps to quickly achieve consensus among the fault-free systems. Gossip protocol is a widely used framework for fault detection services in large-scale cloud systems. Basic implementation lacks in achieving a consistent system state due to random patterns of Gossip messages that result in false failure detection. As well as there is no upper bound on consensus time in random way of gossiping. In this project, we propose to implement the three gossip protocols approach: Round Robin, Binary Round Robin, and Round Robin with Sequence Check[1]. We will also implement Distributed Consensus algorithm to maintain the performance and integrity of the distributed system. We aim to implement and analyze the claims that Gossip provides under different workload systems and cases of failure - single, simultaneous and node rejoining the cluster. These experiments were performed with specific values of configuration parameters - T_{gossip} ,$T_{heartbeat}$, T_{fail} as observed in [3].

Keywords: Distributed Systems, Failure detection, Fault tolerance, Gossip Protocols, Consensus algorithm

ACM Reference Format:
Ritesh Narendra Ghorse, Shreyas Muralidhara, and Tanvi Pandit. 2021. Failure Detection and Consensus in Large-Scale Cloud Systems with Gossip Protocols. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 Introduction

The rapid increase in popularity of parallel systems using off-the-shelf commodity hardware is because of its cost-effectiveness and performance. But it comes with a trade-off that it can fail anytime. So we are required to monitor and manage replication, fault tolerance, load balancing in these systems.[1] It is, however, not as easy as it seems. There are several complexities that we have to deal with such as heterogeneity of technologies in network, different server architectures, underlying operating systems, etc. [10]

Fault tolerance in a distributed system is equally as important as delivering correct results. To provide this, the system requires efficient fault-detection techniques that would reduce the probability of failure of the entire system due to the failure of one single component. The recovery time for a smaller system is relatively quick due to the lesser complexity of the system. However, recovery time in larger systems is significantly higher due to the presence of a complex network of machines which makes the task of even finding the faults too convoluted. In addition to that, we have to implement consensus in a scalable fashion so that a common agreement is reached as early as possible. Traditional group communication methods perform well only for small system sizes and rely on the existence of a broadcast medium[1]

In a large distributed system, the network delays can be long and so it is necessary to detect the failures asynchronously. A failure detector must be able to differentiate between a dead and slow process that is alive[12]. In a peer-to-peer network, all the nodes in the system communicate with each other to achieve consensus about the system state. Consensus can be achieved through various ways, but group communications and gossiping have shown better performance. As we know the Internet is the best try, these methods as well relax the constraints of absolute consensus because if not done, then the implementation of these methods will be complex and with little guarantee that it can work in a finite amount of time in large networks like the Internet. Group communication uses a reliable broadcast or multicast approach to communicate with other members of the system. With gossip, nodes individually gossip about their state and also the state of other nodes with another node, which eventually helps in obtaining the global view across

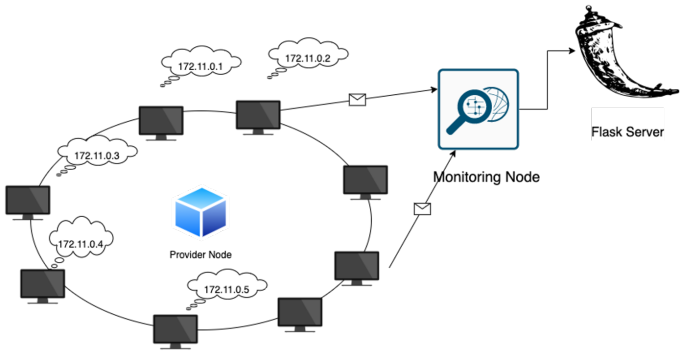

Figure 1. System Architecture

the system.[11]

In this project, we will be using gossip protocol for fault detection. This interest is because of several pros of gossip-style detectors like it does not depend on any single central node or message, it makes minimum assumptions about the network. Random gossiping is by far the most common implementation of the gossip protocol. Random gossiping does have a few drawbacks. Firstly, it may falsely detect that a node is faulty if it has not received any recent heartbeat, secondly, it may also result in the network bandwidth being wasted due to redundancy. This random gossip protocol also does not address how consensus is maintained and how consistency is maintained [1]. We will be implementing an optimization algorithm for gossip with 3 new protocols namely Round Robin gossip protocol, Binary Round Robin, and Round Robin with sequence check. We will also be implementing a distributed consensus algorithm which will be used for agreement about the failing nodes in the network. This algorithm will terminate when all the nodes have come to a common consensus about the failed node.

The remainder of this paper is organized as follows. Section 2 will discuss the design and algorithms used in the paper. Section 2 talks about the architecture design and algorithms that were written. Section 3 mentions about the evaluation bed and results that we have obtained so far.Section 4 concludes the results of our findings and Section 5 will talk about the related works that have been done so far in this domain.

2 Design and Algorithms

We have implemented the following algorithms: Round-Robin (RR), Round-Robin with Sequence Check (RRSC), and Binary Round-Robin (BRR) in Gossip protocol. These algorithms are shown to have a better performance than random gossiping [1]. To complement this, we implement a consensus algorithm to reach an agreeable state as quickly as possible.

2.1 System Architecture

The system used for testing consisted of a virtual private network where each new node joins into the system. The network has a monitoring node attached to it, the sole purpose of this monitoring node is to display all the communication going inside the node. Along with the monitor node, we have a Flask application which retrieves the data from the monitoring node and displays it to the user in the browser. When a node joins into the system, each of those nodes have their own different host names and IP addresses which simulates a real world distributed system. These nodes communicate with each other using XML RPC based protocol. The system architecture is shown in fig 1.

2.2 Algorithms

The objective of our system is simply to keep a log of each active member in the system through gossiping. The log maintained is known as the gossip digest list, it contains a 'heartbeat value', 'generation', and 'application version' of each member - used to keep a track of liveliness of the members. In random gossip protocol, a node randomly sends new digest list to another randomly selected node in the system after every T_{gossip} seconds. The heartbeat counter is incremented after every $T_{heartbeat}$ seconds. If a node in the system does not receive a new heartbeat count from another node in its list for T_{fail} seconds then that node is considered to be failed. But this information is not removed from its list until $T_{cleanup}$ seconds. Thus, after multiple rounds, every node in the system gets information about every other node in the system.

The process of maintaining the node information is same across all the round robin protocols, but they differ in the selection of node to which the node should disseminate. Depending on the modes of selection, we have 3 protocols as described in 2.2.1, 2.2.2, 2.2.3.

2.2.1 Round Robin Protocol. In round-robin gossip, nodes send gossip messages every T_{gossip} seconds in a fixed sequence of rounds. So in a single round, each active node will send and receive a message. The destination node id is determined using the equation:

$$DestNodeID = SrcNodeID + r, 1 \leq r < n$$

where n is the nodes in cluster and r is round number.

This would guarantee that each node in the system receives gossip updates in a finite number of rounds. In this way, the consensus in an n-round round-robin system would require:

$$T_{cleanup} \geq \alpha T_{gossip}$$

The communication between the nodes in a round robin protocol is illustrated here in fig 2 [1]

In round robin protocol, following are the sequence of message exchange. In the first round, node 1 will send it's gossip to node 2, in the second round it will send to node 3 and in the third round will send it to node 4. In the second round, node 2 will have updated information about node 1(from the first round) which it will send to node 4 in round 3. Thus, in fourth round, node 1 will send an information to node 4. Since node 1 has incremented its heartbeat in the meantime, node 4 will update itself about node 1. Thus, new information keeps on converging across the cluster.

In order to optimize the node selection and reduce the time for convergence, we have next protocol that is Binary round robin gossip protocol.

2.2.2 Binary Round Robin Protocol. This is similar to round-robin gossip but uses a different equation to calculate destination node id to reduce gossiping rounds and hence the convergence time across the cluster. It uses the following equation:

$$DestNodeID = SrcNodeID + 2^{r-1}, 1 \leq r < log_2(n)$$

So a consensus in an n-round round-robin system would require :

$$T_{cleanup} \geq (log_2(n))T_{gossip}$$

Thus, comparison for the no of rounds required for Round Robin and Binary Round Robin is given in table 1 [1]

Number of Node	2	4	8	16	32
Round Robin	1	2	4	5	8
Binary Round Robin	1	2	3	4	5

Table 1. Number of rounds per system size

Fig 3 [1] illustrates the relationship between round count and node count on 8-node binary round-robin system.

2.2.3 Round-Robin with Sequence Check. This approach requires a slight modification in the round-robin gossiping. They are:

1. If a node doesn't receive a gossip from an expected source node in a particular round, then that node is suspected to be failed at the end of that round.
2. A failure node is marked as alive if and only if a gossip digest list is received from that node itself.

2.2.4 Consensus Algorithm. Early failure detection techniques work on basic timeout mechanism. However, it is possible that such services are prone to network failure, delays and message losses. Gossip style failure detectors, though fully distributed is not immune to false failure detection specifically with random gossip patterns. In order to obtain a consistent system view and prevent false failure detection, it is necessary for all the nodes in the system to come to a consensus on the status of a failed node.

RAM	8GB
Processor	Intel i5
Cores	4 cores

Table 2. Testing System Specification

The consensus algorithm requires each node to maintain their own view regarding the status of the nodes in the system as well as monitor the suspicions of all the other active nodes. For this, each node maintains a bit vector of size n, where each jth bit refers to the jth node in the system. In every round, a node shares this fault-vector with the gossiping node. Thus, each node maintains its own fault-matrix and keeps the values updated with every gossip round. Whenever a destination node suspects that any other source node in the system may have failed, it checks the corresponding column of its suspect matrix to check with what other nodes have to say about source node. If all of the nodes suspect that a source node has failed, then that node is not included in consensus, that is, its opinion is discarded. However, the destination node does not remove the suspected node immediately but waits for $T_{cleanup}$ seconds. If the node does not hear back about that suspected node even after $T_{cleanup}$ second, then it marks that node as failed and this information is broadcasted to all the nodes in successive gossip rounds.

Figure 4 illustrates working of consensus algorithm in a 4-node system with one failed node, node 0. Node 2 sends a suspect matrix to node 1 indicating that itself and node 3 suspect node 0 may have failed based in part upon earlier messages received from node 2-3. Node 1, which already suspects node 0, updates its suspect matrix on receipt of a gossip message from node 2 and finds that every other node in the system suspects that node 0 has failed.

Thus, a consensus has been reached on the failure of node 0 and this information is broadcasted throughout the system in regular gossip rounds. All the nodes subsequently update their live lists to indicate the status of node 0.

3 Experiment Evaluation

3.1 Experiment Setup

The experiments were conducted on a single cluster running on a local host system. The specifications of the host system are specified in Table 6

The four protocols were implemented as mentioned above and test cases were designed for failure detection. The cases were as follows

1. Test 1 : Single Node Failure
2. Test 2 : Simultaneous Node Failure (2 Nodes)
3. Test 3 : Dead Node Becomes Alive

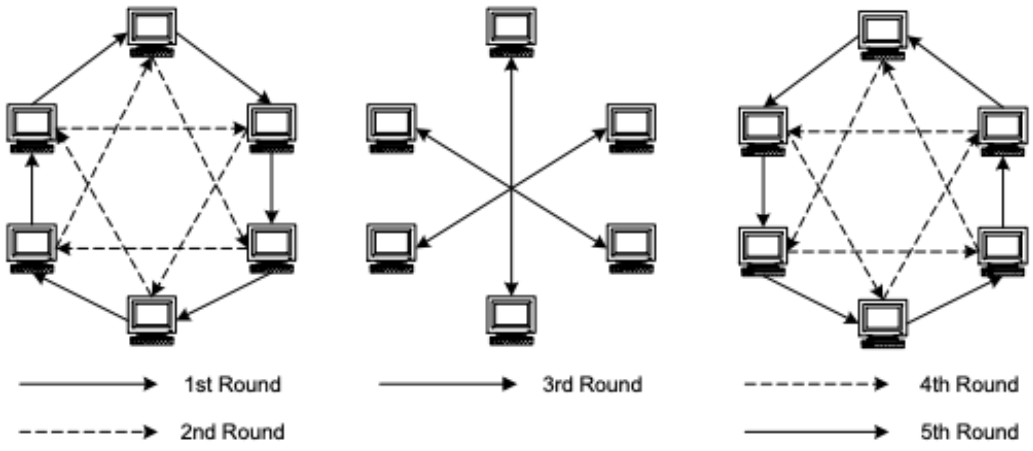

Figure 2. Communication in Round Robin protocol

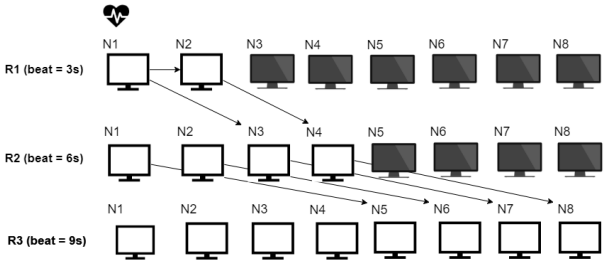

Figure 3. Gossip distribution in Binary Round Robin protocol

For each of these test cases, we created a system with 8, 16 and 32 nodes to measure our results. With every test case, we were recording the nodes once it has reached a stable state. Stable state in our case is when all the nodes have completed their initial handshake. This system was setup using a virtual private network on an i5 Quad core 2.5 GHz machine. Whenever there is any change to the network, the nodes gossip and comes to a consensus about the state of the system, this consensus results were seen on the flask server through monitor-node. These consensus results consists of information about the average time to reach consensus, no of messaged exchanged as well as no of false failure detection. For each test case and for each node setup, we took a series of 10 readings and averaged the values before plotting the results.The average and standard deviations for each test cases are mentioned in the table.

3.2 Results

The results were derived keeping two assumptions in mind,

1. There always exists atleast 2 active nodes in the system
2. The provider node never goes down.

For this experiment setup we have kept gossip time T_{gossip} = 3 seconds and $T_{cleanup}$ = 11 seconds. The size of the suspect matrices increases as O(n2) with the system size resulting in larger computation time for consensus. For the resources described in the system specification, we performed experiments until 32 nodes. We have compared variation in consensus time with increase in the no of nodes for all the 4 protocols in each test case.

Test Case 1 : Single Node Failure Figure 5 shows the variation in consensus time as the no of nodes increases. From the graph, we see that in case of Binary Round Robin, the consensus time decreases with increase in the number of nodes. The reason for this steep decrease is when we increase the no of nodes, more and more number of nodes transmit the same information to their peers in as single round and hence it becomes faster to achieve consensus (Figure 3. Sequence checked RR looks like a more stable option since it's value is lower in the initial case for smaller number of nodes and it becomes stable as the number of nodes increases. For random gossip, we see the consensus time is almost constant with respect to increase in no of nodes, it would be interesting to see how random scales with increasing no of nodes. We limited the testing to 32 nodes since increasing the nodes would not work well on our local system
Thus to conclude, we see binary round robin and Sequence Check RR to be the promising options for single node failure along with random gossip.

Test Case 2: Simultaneous node failure For this experiment, we failed 2 nodes simultaneously for different number

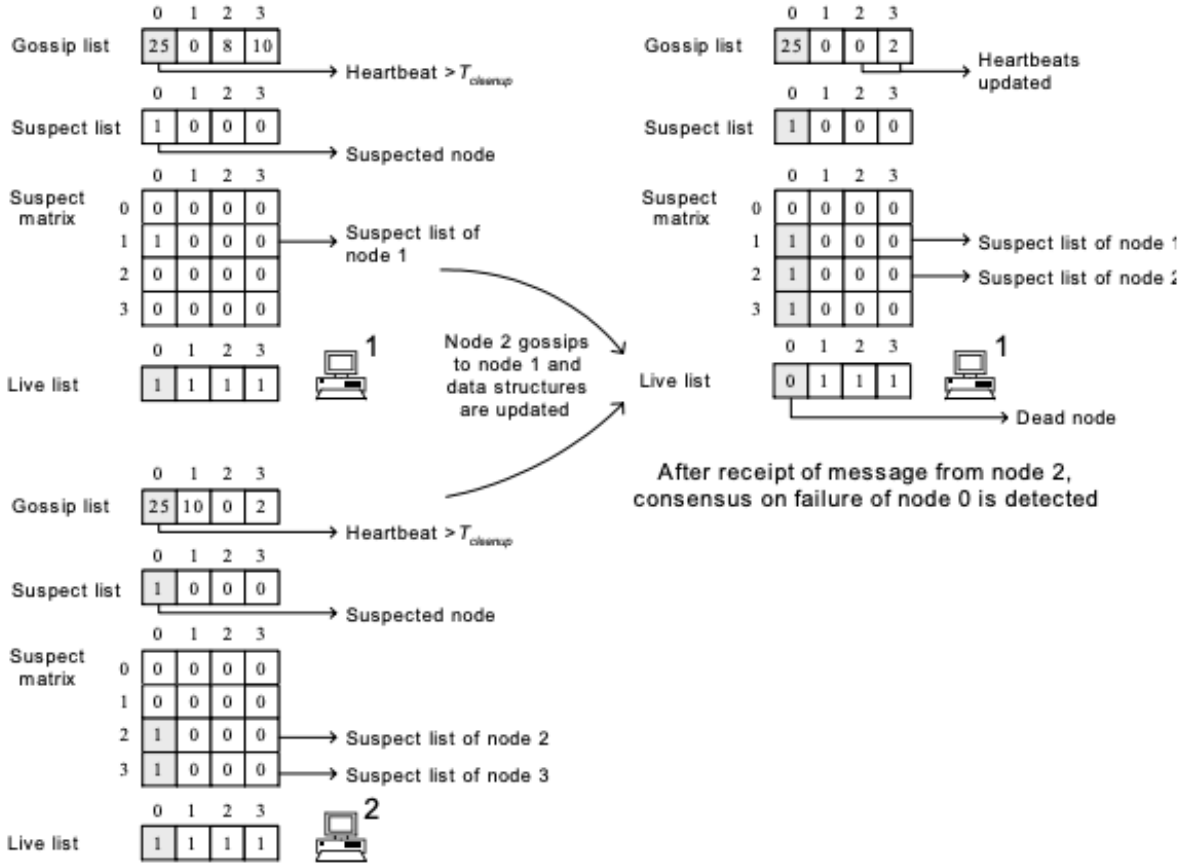

Figure 4. Consensus algorithm in a 4 node system

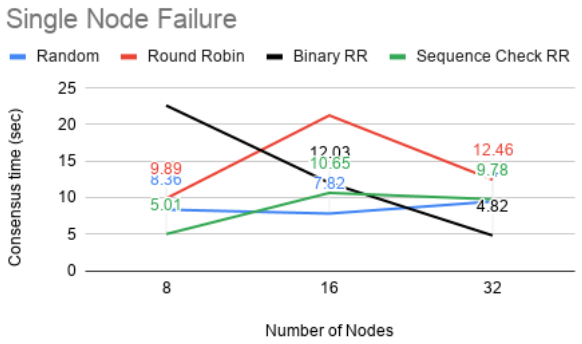

Figure 5. Case 1: Single Node Failure Detection across protocols

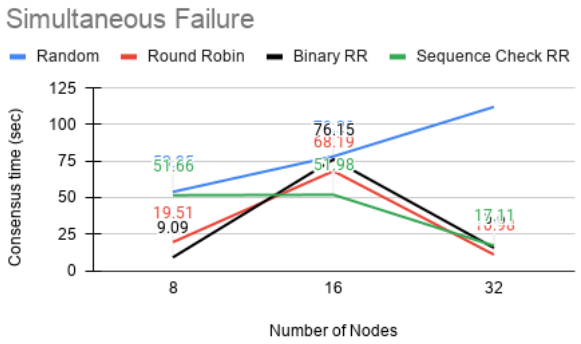

Figure 6. Case 2: Simultaneous Failure detection across protocols

of nodes. The result of this experiment is seen in fig 6. As we know random does not have a fixed gossip pattern and so when multiple nodes fail at the same time, it will take longer to achieve consensus. This same trend is observed from figure 6. We see a triangular trend for Round robin and binary round robin. But here binary round robin has a flaw that it updated an information about a second node indirectly from a third node. Because we have less number of nodes, the proportion of nodes that knows the truth to the nodes that bluff is less, hence it takes longer time for all the nodes to come to consensus. This contrast with the behavior

of binary round robin in single node failure because this proportion may be even lesser and hence it takes more time as compared simultaneous node failure in case of binary round robin. The way sequence check is different from other round robin protocols is that the RRSC protocol suspects failure after every round. But again it has to compute the consensus more frequently than other protocols which consumes extra time.

Here again we see Sequence checked RR to be the winner for greater number of nodes whereas random did not scale well with increase in no of nodes in the cluster.

Test Case 3: Dead Node becomes Alive The figure 7 shows the results when dead node in the system becomes alive. This case deals more about each node being appearing down for some time because of either transmission delay, slow processing, etc than the node leaving the networking and then joining back. In this case, we see that ordered protocols work better than random gossip. Amongst the ordered protocols,binary and sequence check round robin seems to perform better because once the node is alive and one nodes gets to know about this and passes this information to other node which subsequently passes this information in further rounds, it converges faster. So, in this test scenario we see binary round robin and Sequence checked round robin emerges as winner.

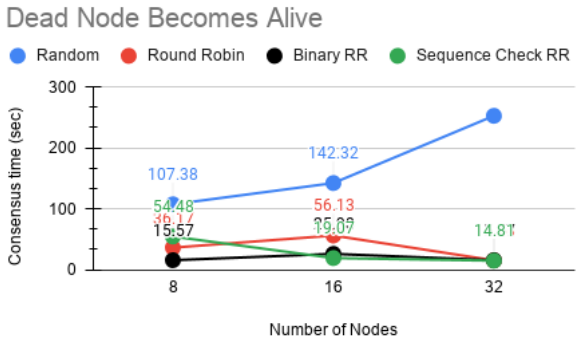

Figure 7. Case 3: Restarting dead node across protocols

Thus, from all the test cases we get to understand that Random gossip, binary round robin and sequence checked round robin are promising options. However, to come to a conclusion we need to ascertain how each of the test cases perform across nodes for all the algorithms.

Scalability of Random Gossip : Comparing the overall performance of Random gossip across nodes, we come to a conclusion that random gossip scales linearly as the number of nodes in the system increases in all test cases. This makes random gossip a good candidate for systems which either have less number of nodes so that the consensus time can be kept to a low minimum. Also this protocol is fairly simple and easy to implement. Fig 8 shows the plot of Random gossip across a number of nodes.

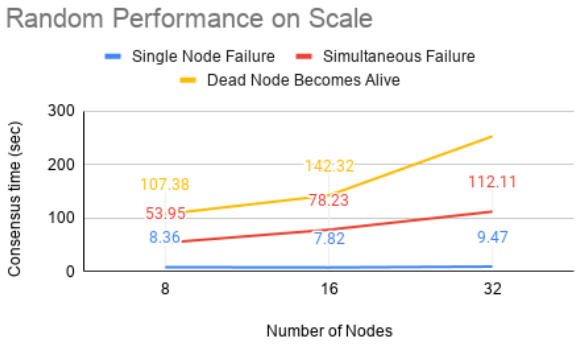

Figure 8. Random Performance on Scale

Scalability of Round Robin gossip protocols: Fig 10, 11, 9 shows the scalability of Round robin for various events. We see that the consensus times increases to a maximum and then begins to decrease. This trend is constant across all the variants of round robin protocols. Thus, from these plots

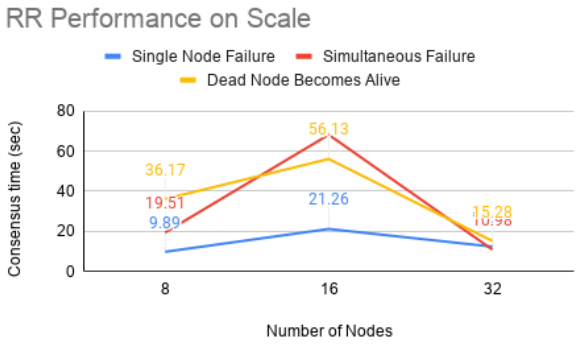

Figure 9. Round Robin Performance on Scale

we can infer that Binary Round robin and Sequence checked round robin protocols scale well across the nodes. However, in one caveat in case of Binary round robin for it to scale well is that the no of nodes should be in exponents of 2. If it is not in exponents of 2, then there is serious performance degradation. Thus, Sequence checked gossip protocol is the most scalable option. Also binary round robin can produce false failure in third test case as it doesn't have that extra constraint as given in the sequence check round robin.

4 Conclusion

In this project, we implemented several gossip-style, failure-detection protocols which are an advancement to the existing random gossip protocol in a single layered architecture. Several performance experiments are conducted to study the

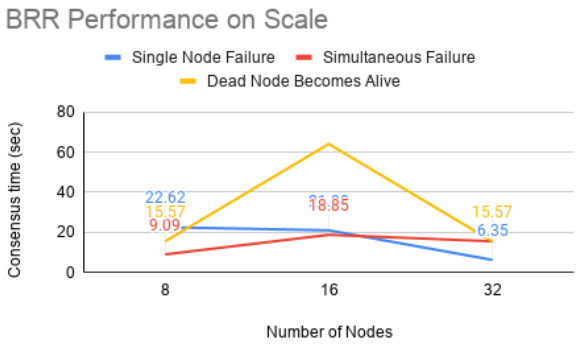

Figure 10. Binary Round Robin Performance on Scale

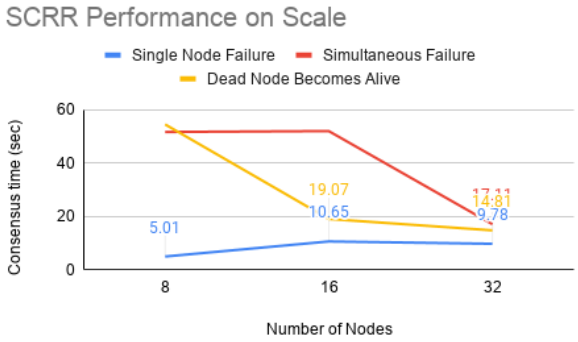

Figure 11. Sequence Check Round Robin Performance on Scale

Test Case	8	16	32
Single Node Failure	0.72	0.69	1.31
Simultaneous Failure	1.71	1.21	1.62
Dead Node Becomes Failure	1.33	1.49	1.47

Table 3. Standard Deviation for Random Gossip (10 runs)

Test Case	8	16	32
Single Node Failure	0.77	1.33	0.71
Simultaneous Failure	1.27	1.91	1.82
Dead Node Becomes Failure	1.63	1.43	1.61

Table 4. Standard Deviation for Round Robin Gossip (10 runs)

characteristics of these gossip and consensus techniques in terms of consensus time and scalability.

Our experimental studies indicate that the round robin gossip protocols provide the means for implementing an efficient failure-detection service with consensus for large-scale clusters. When compared to our base random gossip protocol, the round-robin protocol has fewer false failure

Test Case	8	16	32
Single Node Failure	1.33	4.72	1.91
Simultaneous Failure	1.21	2.59	2.27
Dead Node Becomes Failure	1.83	3.61	3.83

Table 5. Standard Deviation for Binary Round Robin Gossip (10 runs)

Test Case	8	16	32
Single Node Failure	0.69	1.21	1.33
Simultaneous Failure	1.31	1.63	1.11
Dead Node Becomes Failure	1.18	1.88	1.29

Table 6. Standard Deviation for Sequence Check Round Robin Gossip (10 runs)

detections because of a more efficient and deterministic gossiping order. One benefit of the deterministic gossiping order is that, for a fixed value of T_{gossip}, lower values of $T_{cleanup}$ can be achieved, thereby lowering the response time needed to arrive at consensus. The round-robin protocol with sequence check is a resilient protocol that yields low failure-detection times.

In the analysis presented, the consensus time is the time required for reaching consensus among the nodes in the cluster. Our studies on the random gossiping protocols with consensus reveal the poor scalability of a single layered implementation of the consensus algorithm. However, Round robin with sequence check emerges as the most viable option for system with higher no of nodes. A good outcome would be to use a hybrid approach where we utilize the advantage of Random round robin i.e. better performance with small no of nodes and Sequence Check RR i.e. good performance at scale.

5 Related Work

Gossip is initially presented as simple protocol for failure detection service and to optimize the performance in face of failure. This implementation assumes uniform network topology with bounded number of host crashes [4]. But the proposed approach used random gossip protocol that failed to detect the failures in asynchronous manner. This drawback was overcome by Gossip-style protocols and consensus algorithm[1], which forms the base of our implementation.

To better utilize the gossip protocol for large-scale heterogeneous cluster, GEMS [5] proposes to divide systems into groups of node and layers of communication for fast response time and lower utilization. This idea is further extended by a multi-tier architecture for layered gossip protocol that aggregates the monitoring information related to resource usage for better lookup and identification of redundant capacity [2]. Implementing the gossip protocol with fault and layered approach on a experimental testbed and analysing the same

in terms of consensus time and scalability is demonstrated by [3].

Further motivation was by Apache Cassandra Gossiper implementation[7] as an API with application state object wrapped in key/value pair which provides separate methods for state - Join, Alive, dead and On change. It also explains the data structures to be considered for implementation. Additional function implementations in Redis Cluster[8] through gossip protocols for health detection and fail over, Status update and conflict resolution and Handshake Coupling is good reference for functional implementation. It also presents protocol analysis along with cluster message and structure of source code which helps understand cluster implementation for Gossip.

References

[1] Ranganathan, S., George, A.D., Todd, R.W. et al. Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters. Cluster Computing 4, 197–209 (2001). https://doi.org/10.1023/A:1011494323443

[2] Ward, J. S. and A. Barker. "Monitoring Large-Scale Cloud Systems with Layered Gossip Protocols." ArXiv abs/1305.7403 (2013): n. pag.

[3] George, Alan & Sistla, Krishnakanth & Todd, Robert & Tilak, Raghukul. (2001). Performance Analysis of Flat and Layered Gossip Services for Failure Detection and Consensus in Scalable Heterogeneous Clusters.. 84. 10.1109/IPDPS.2001.925035.

[4] Robbert van Renesse, Yaron Minsky, and Mark Hayden. 2009. A gossip-style failure detection service. In Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98). Springer-Verlag, Berlin, Heidelberg, 55–70.

[5] Subramaniyan, Rajagopal & Raman, Pirabhu & George, Alan & Radlinski, Matthew. (2006). GEMS: Gossip-Enabled Monitoring Service for Scalable Heterogeneous Distributed Systems. Cluster Computing. 9. 101-120. 10.1007/s10586-006-4900-5.

[6] Huang, Shing-Tsaan. "Termination detection by using distributed snapshots." Inf. Process. Lett. 32 (1989): 113-119.

[7] Cassandra Gossip Architecture - https://cwiki.apache.org/confluence/display/CASSANDRA2/ArchitectureGossip.

[8] ApsaraDB Redis cluster Gossip protocol - https://www.alibabacloud.com/blog/in-depth-analysis-of-redis-cluster-gossip-protocol_594706.

[9] Michael Chow and Robbert van Renesse. 2010. A middleware for gossip protocols. In Proceedings of the 9th international conference on Peer-to-peer systems (IPTPS'10). USENIX Association, USA, 8.

[10] Sistla, Krishnakanth & George, Alan & Todd, Robert. (2003). Experimental Analysis of a Gossip-Based Service for Scalable, Distributed Failure Detection and Consensus. Cluster Computing. 6. 237-251. 10.1023/A:1023592621046.

[11] Burns, Mark & George, Alan & Wallace, Bradley. (1999). Simulative performance analysis of gossip failure detection for scalable distributed systems. Cluster Computing. 2. 207-217. 10.1023/A:1019086910915.

[12] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. 1996. The weakest failure detector for solving consensus. J. ACM 43, 4 (July 1996), 685–722. DOI:https://doi.org/10.1145/234533.234549