Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters

Sridharan Ranganathan, Alan D. George,
Robert W. Todd, and Matthew C. Chidester

High-performance Computing and Simulation (HCS) Research Laboratory
Department of Electrical and Computer Engineering, University of Florida
P.O. Box 116200, Gainesville, FL 32611-6200

Abstract — Gossip protocols provide a means by which failures can be detected in large, distributed systems in an asynchronous manner without the limits associated with reliable multicasting for group communications. However, in order to be effective with application recovery and reconfiguration, these protocols require mechanisms by which failures can be detected with system-wide consensus in a scalable fashion. This paper presents three new gossip-style protocols supported by a novel algorithm to achieve consensus in scalable, heterogeneous clusters. The round-robin protocol improves on basic randomized gossiping by distributing gossip messages in a deterministic order that optimizes bandwidth consumption. Redundant gossiping is completely eliminated in the binary round-robin protocol, and the round-robin with sequence check protocol is a useful extension that yields efficient detection times without the need for system-specific optimization. The distributed consensus algorithm works with these gossip protocols to achieve agreement among the operable nodes in the cluster on the state of the system featuring either a flat or a layered design. The various protocols are simulated and evaluated in terms of consensus time and scalability using a high-fidelity, fault-injection model for distributed systems comprised of clusters of workstations connected by high-performance networks.

Index Terms — Cluster computing, consensus, failure detection, fault tolerance, gossip protocol, Myrinet.

1 INTRODUCTION

Loosely coupled clusters of workstations, connected together by high-speed networks for parallel applications, have increased in popularity due to greater cost-effectiveness and performance. Such clusters are attractive as highly available and scalable supercomputing resources. In order to build inexpensive systems from commercial off-the-shelf components, diverse hardware and software from various vendors need to be incorporated. The task of managing these heterogeneous clusters as a single unified resource involves replication, load balancing, and other distributed services.

Fault tolerance is an important service requiring efficient detection techniques that minimize the effect of failures on the applications that execute in a large system. Services with small failure detection times allow the applications to recover and reconfigure quickly. However, minimizing the failure detection time is a non-trivial problem because the process of detecting faults in such systems is complex. The task is compounded by the need to implement consensus in a scalable fashion. Traditional group-communication methods perform well only for small system sizes and rely on the existence of a broadcast medium.

Random gossiping, which is based on the individual exchange of ‘liveliness’ information between nodes, has been investigated as a possible failure-detection approach for scalable heterogeneous systems [5,17]. This interest is due to several advantages of gossip-style failure detectors. For example, the algorithm is resilient and does not critically depend upon any single node or message. Moreover, the protocol makes minimal assumptions about the network.

This paper describes a distributed consensus protocol that enables nodes to come to an agreement about dynamic node failures within the system. This distributed mechanism for failure detection completes successfully when all fault-free processors independently diagnose the state of the system and come to a consensus on the correct system state. This paper investigates optimizations of existing gossiping algorithms in the form of three new protocols: Round-Robin (RR), Round-Robin with Sequence Check (RRSC) and Binary Round-Robin (BRR) incorporated with this consensus algorithm.

The remainder of this paper is organized as follows. The next section outlines the design and describes the parameters that control the performance of existing gossip-style failure detection protocols. The third section describes existing methods and models used for solving consensus in distributed systems. The design of the new protocols and the consensus algorithm is explained in the fourth section while section five provides a simulative analysis of the new gossiping protocols. Finally, the sixth section offers conclusions to the work presented and suggests future directions for work in this area.

2 RELATED RESEARCH

In their 1972 paper, Baker and Shostak describe a gossip protocol using ladies and telephones as an example [16]. Gossiping has also been investigated for replicating databases in the form of epidemic algorithms [12]. The protocol itself is based on the Clearinghouse project that pioneered work in this direction for solving data inconsistencies [18].

Van Renesse, Minsky and Haden first investigated gossiping for failure detection [17]. In this paper, they present three protocols: a basic version, a hierarchical version, and a protocol that deals with arbitrary host failures and partitions. All protocols have the simple objective of trying to determine which other nodes in the system are gossiping by keeping a log of each known member. The log, known as a gossip list, also contains the ‘heartbeat value’ of each member – an arbitrary integer that is used to keep track of the liveliness of the members. Every T_{gossip} seconds, each node updates its gossip list by incrementing its heartbeat value and gossips the updated list to an arbitrarily chosen node. The target node then merges the received list with its own by adopting the maximum heartbeat value from the two lists for each member. Each node monitors the gossip list to check for liveliness of all the other nodes in the system. If the heartbeat value of a particular member has not increased for a period of more than T_{fail} seconds, the member is suspected to have failed and is marked as such. The member is then removed from the list if it continues to be unresponsive for a duration of $T_{cleanup}$ seconds.

Burns, George and Wallace conducted a study that focused on a simulative performance analysis of several gossip protocols operating on a distributed systems model comprised of clusters of nodes connected by Myrinet [5]. They employed a modified version of the gossip protocol in which members were removed from the list after $T_{cleanup}$ seconds without the need for a check at T_{fail} seconds, a convention adopted in this paper. Their modification simplifies the basic gossip protocol, requiring the configuration of only two parameters, T_{gossip} and $T_{cleanup}$. Although the basic protocol is sufficient in a small system, it was found to yield poor scalability with increases in system size. The inefficiency of the basic gossiping protocol stems from the fact that a large amount of redundant information is sent between clusters where the communication penalty is typically larger than for intra-cluster communication. Localizing information dissemination can eliminate this penalty. The hierarchical protocol makes use of *a priori* knowledge of the underlying network topology to send a majority of the gossips locally. The local gossip factor, L, can be configured for the particular network to control the level of inter-cluster gossips. A *piggyback* protocol was introduced that further reduces the network bandwidth required for gossiping by monitoring traffic generation patterns and piggybacking gossips on application-generated messages whenever possible. The piggyback factor P specifies a period in which gossips cannot be augmented onto existing traffic and can be configured to control the inter-gossip interval and consequently the amount of bandwidth used. In their simulative study of the three protocols, both the hierarchical and piggyback protocols were found to be significantly more scalable than the basic protocol for gossiping. However, the practicality of implementing the piggyback protocol on top of user applications requires the management of several threads of control. Additionally, the efficiency of the protocol depends upon the rate at which the application messages are generated on which gossips can be piggybacked. If an application does not generate messages at a sufficient frequency, the piggyback protocol degenerates to the hierarchical protocol, as it is required to self-generate gossip messages.

In previous studies the performance of gossip protocols has been evaluated based on the time taken for all processors to detect a fault. However, largely missing from previous studies is the issue of reaching *consensus* throughout the system on the *state* of the system, and the algorithm and performance impact in doing so. The following section provides an overview of the models and methods that may be employed for attaining consensus in distributed systems.

3 THE CONSENSUS PROBLEM

The primary goal of clustering independent computer components is to create a single unified resource with greater performance and availability than its individual parts. In order to achieve availability by identifying the fault-free resources in the system, each component in the system must reach an agreement on the state of other components in the system – also known as solving the consensus problem. Formally stated, the consensus problem is defined as the process by which agreement is reached among the fault-free processors on a quantum of information in order to maintain the performance and integrity of the system [3]. The components within the system may require agreement on issues such as global clock synchronization, resolution of the contents of messages, system configuration protocols, and database commit protocols.

Several situations that can lead to deadlock of a system implementing the consensus algorithm have been identified. Turek and Shasha provide a brief overview of some of these results [14]. Their study showed that the high-performance interconnect used within the system plays an important role in solving the consensus problem. For example, consensus can never be reached if the network delay is unbounded and there is no guaranteed delivery of messages. Several consensus impossibility theorems can be found in the literature [2,9,10,14]. It has also been shown that consensus is possible only in a synchronous system with bounded message delay or if ordered atomic broadcast is available. Formally, a correct consensus protocol is [3]:

- **Consistent:** All processors agree on the same value and all decisions are final,
- **Valid:** The agreed upon value was the initial input of some agent,
- **Wait-free:** All processors must reach agreement in a finite number of steps.

The traditional method for implementing consensus is N-Modular Redundancy (NMR). With NMR, n processors perform the same task and, by taking a majority vote, mask at most t faults, where $n \geq 2t+1$. This process is simple but expensive since the throughput of the entire system is limited to that of the slowest individual component. The need for a centralized operating system that is aware of the condition of the active processing elements led to the development of system diagnosis methods like the PMC model [11]. In the Preparata, Metz, and Chien model, processors perform tests on one another in a predefined sequence and a centralized arbiter uses the results of all the tests to diagnose the entire system. One drawback in this model is the assumption about the existence of tests that provide full coverage. The ultra-reliable arbiter requires that all test data be gathered in a central node and analyzed, with the result distributed back to the system. The collection and redistribution of state information within the system can be costly in terms of time and message-passing bandwidth. Clearly, the centralized supervisor is a bottleneck that requires the development of more scalable methods such as distributed diagnosis.

Distributed diagnosis requires that each fault-free processor independently diagnoses the entire system. The independent diagnosis performed is then relayed to the user via one of several mechanisms [3]:

- **Information dissemination**: The results of the diagnosis are propagated on the network, so that any processing element may be queried to learn the state of the entire system,
- **Message passing**: The user learns the state of the system by querying all the fault-free processors via message passing,
- **Roving diagnosis**: Diagnosis at any given time is performed only by a subset of the nodes that roves over the entire system.

Kuhl and Reddy [8] proposed a series of distributed diagnosis algorithms called SELF in which each processor P_i maintains a fault vector $\boldsymbol{F_i}$ whose j^{th} element is zero if processor P_i concludes that P_j is fault-free and one otherwise. A processor P_i tests each of its neighbors and updates the fault vector. If a neighbor P_j is found to be faulty, the fault vector is broadcast to all the other neighboring fault-free processors. The other fault-free processors confirm the state of P_j by testing both P_i and P_j and broadcast the results to their neighboring fault-free processors.

However, the performance of the SELF algorithm depends heavily on the effectiveness of each test. The algorithm has limited scalability in that it assumes an efficient broadcast mechanism. Several modifications and optimizations of the SELF algorithm have been presented that improve the performance and adaptability to different network topologies. Though the new consensus protocol presented in this paper is loosely based on the SELF algorithm, a gossip-style technique for processor membership is used in place of a distributed diagnosis approach.

Processor membership methods are an alternative set of approaches that aim at managing large-scale clusters as a single unified resource. The methods are similar to distributed diagnosis methods in that each processor must independently diagnose the system without the aid of a centralized arbiter. However, instead of testing neighboring processors, fault detection is performed by monitoring periodic alive messages that are transmitted on the network by each fault-free processor. Examples of processor membership methods are [3]:

- **Periodic broadcast**: Designed for synchronous systems with atomic broadcast yielding very small failure detection times. However, the network bandwidth consumed by each process can be prohibitive.
- **Attendance list**: Processors take turns to initiate a roll call and to check the timeliness of the attendance list.
- **Neighbor surveillance**: Nodes request confirmations from their neighbors in a pre-defined logical cycle.
- **Gossiping**: Individual exchange of liveliness information is carried out simultaneously between all nodes in the system.

The last three methods have slower detection times but consume less bandwidth and do not require the network to have a hardware broadcast capability. Recently, gossiping has come to the forefront as a promising method for designing failure-detection services on large-scale systems in a scalable fashion. The following section describes a gossip-style consensus algorithm as well as several novel gossiping protocols.

4 Protocol Design

The basic gossip protocols presented in the previous section suffer from several limitations. One limitation is that the random pattern of gossip messages can result in false failure detections. A node may falsely detect a failure in a second node simply because it has not received any gossip messages containing a recent heartbeat counter for the second node. Even with $T_{cleanup}$ set to a relatively high value, there still exists a small probability for false failure detections. A second limitation to random gossiping is that network bandwidth may be wasted. Multiple nodes may send gossip messages to the same destination in a given T_{gossip} interval while other nodes receive no gossip messages. Finally, the basic gossiping protocols do not address consensus and the importance of maintaining a consistent system view where all fault-free nodes come to an agreement on the identity of the faulty nodes.

This paper addresses the need for new techniques to reduce false failure detections, optimize bandwidth utilization and scalability of failure detection services, and achieve consensus. The *round-robin*, *binary round-robin* and *round-robin with sequence check* protocols proposed in this paper aim to optimize bandwidth consumption while simultaneously addressing the consensus problem.

4.1 The Consensus Algorithm

In the traditional gossip approach, each node independently detects any failures in a system by examining its gossip list. In order to preserve a consistent system view and prevent false failure detections, it is necessary for all the nodes to come to a consensus concerning the status of the failed node. This subsection describes an algorithm in which consensus is reached when each fault-free node in a gossiping system realizes that all the fault-free nodes have detected a failed node. The issues of consensus on node insertion or restoration are beyond the scope of this paper. The remainder of this section focuses on modifying the gossip algorithms to reduce the time required to reach consensus.

The consensus algorithm requires each node to not only maintain its own view regarding the status of the other nodes, but also monitor the suspicions of all other fault-free nodes. To achieve this functionality, each node in a system of n nodes maintains a suspect bit-vector of size n, which is initialized to all zeros. The j^{th} bit of this vector is set when node j is suspected to have failed due to the expiration of its heartbeat value in the gossip list. The suspect vector from each node is shared between all other nodes to yield a two-dimensional suspect matrix $\boldsymbol{S}$ of size n x n. An element $\boldsymbol{S}[i,j]$ is set to one if node i suspects node j to have failed.

Consensus is reached when a given column of $\boldsymbol{S}$ contains a one for all i corresponding to a fault-free node. The fault-free nodes are monitored by maintaining a separate fault vector $\boldsymbol{F}$ at each node. Element j of $\boldsymbol{F}$ is set to one if a majority of the nodes suspect it to be faulty and a logical OR operation between $\boldsymbol{F}$ and $\boldsymbol{S}$ is performed to mask the faulty elements. Therefore, consensus is reached when the result of the OR operation yields a bit array in which all the elements are one. When a node detects that consensus has been reached, it performs the necessary steps required for reconfiguring the system and broadcasts a message to all the other nodes indicating that consensus was achieved. Fig. 1 illustrates the basic steps in the consensus algorithm for a four-node system.

In step 1 of this figure, the suspect matrix in node 1 indicates that it suspects node 0 to have died. Node 2 also suspects node 0 to be dead and has received a suspect matrix from node 3 with the same supposition. In step 2, node 1 gossips its suspect matrix to node 2, which merges the received matrix with its own. At this point, node 2 detects that a majority of the nodes suspect node 0 to have died, so the bit corresponding to node 0 in the fault vector is set in step 3. In step 3, node 2 detects that consensus has been reached since all fault-free nodes, corresponding to zero bits in the fault vector, detect the failure of node 0. In an additional step not shown in the figure, node 2 multicasts a message indicating the attainment of consensus to nodes 1 and 3 and performs operations required for reconfiguring the system. In a larger system, more gossips may be required between the time a node detects a majority and sets the bit in the fault vector and the time when all non-faulty nodes are found to have reached consensus.

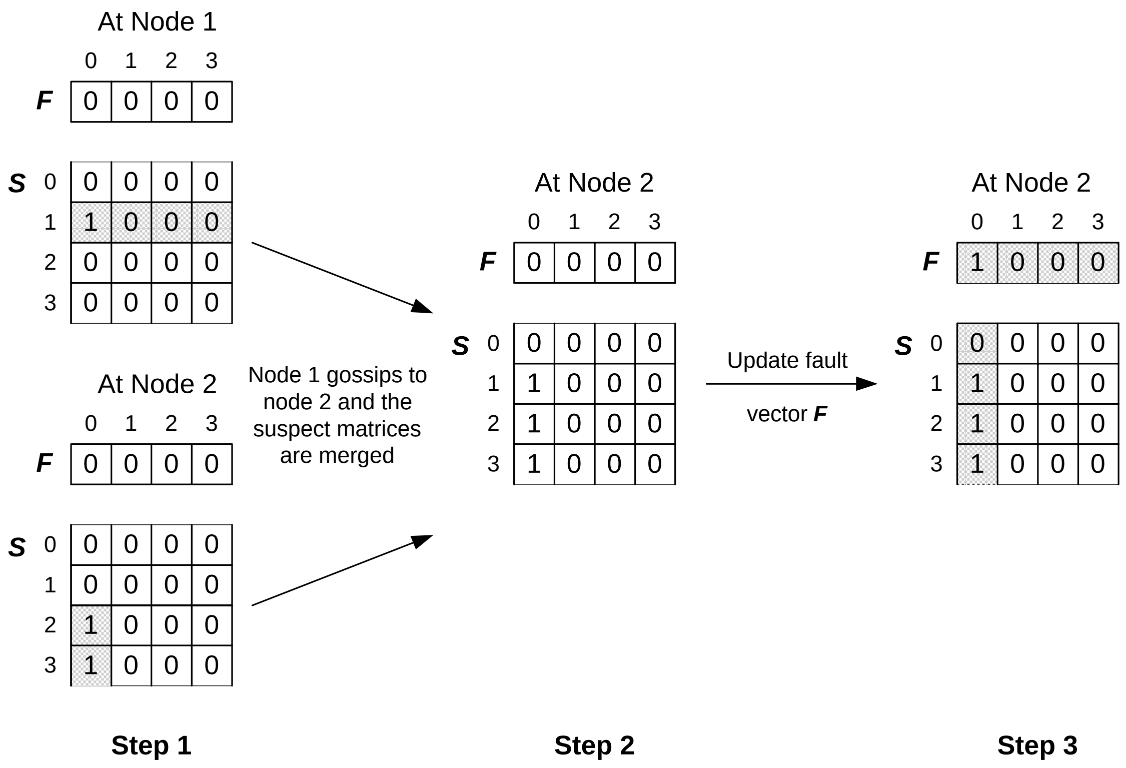

Fig. 1. Illustration of the consensus algorithm on a 4-node system.

A detailed description of the consensus problem is provided in the pseudo-code shown in Fig. 2. This algorithm is executed in each one of the *n* nodes to maintain the fault vector and distribute the suspect matrix. The actual process of failure detection takes place by examining the heartbeats in the gossip list, which is not included in the algorithm. Similarly, other related operations such as the merging of gossip lists and the updating of heartbeats are assumed to be implicit and not included.

The algorithm can be divided into four steps: updating the suspect matrix when a new gossip arrives, updating the node's own row of the suspect matrix before sending a new gossip, checking for a majority of nodes suspecting a failure, and checking for the consensus state. The operations that need to be performed to update the suspect matrix when a gossip message is received are shown in lines 3–16. When node *p* receives a gossip from *q*, the locally stored suspect matrix ***S*** and the recently received suspect matrix ***S'*** are merged by performing a bitwise OR of all the elements *except* the row belonging to node *q*. The reason for this exemption is that it is possible for node *q* to have made a false detection resulting in an entry value of one in the corresponding element of its suspect matrix. When the error is later discovered, the corresponding element is reset to zero. Entry changes from one to zero will be masked by the logical OR operation. Therefore, the q^{th} row of ***S*** should be replaced with that of ***S'*** to remove any false detections that node *q* might have made. The logical OR operation is performed on the rest of the rows as described.

```
1.  for Node ID = p where 0≤ p < n do
2.      Initialize F[i] and S[j, k]  to 0 where  0≤  i , j , k< n
3.      On the receipt of S' from Node q do                  // Update S when another node
4.          for (j=0, j<n, j++)                               // gossips its suspect matrix S'
5.              if (j≠q) then
6.                  for (k=0, k<n, k++)
7.                      S[j, k]=S[ j, k] || S'[j, k]          // Logically OR the elements
8.                  end for                                   // for all rows except the qth
9.              end if                                        // row
10.             if (j=q) then
11.                 for (k=0, k<n, k++)
12.                     S[j, k]=S'[j, k]                      // The qth row should be
13.                 end for                                   // replaced with the qth row of S'
14.             end if
15.         end for
16.     end do
17.     Every $T_{gossip}$ seconds do                         // Update S and F before
18.         for (k=0, k<n, k++)                               // gossiping; Node ID = p
19.             if ( fail_check(k)= true) then                // fail_check is a
20.                 S[p,k]← 1 else S[p,k]← 0                  // procedure to check
21.         end for                                           // for liveliness
22.         for (k=0, k<n, k++)
23.             temp← 0
24.             for (j=0, j<n, j++)
25.                 if  S[j, k]=1  then temp← temp+1          // Set F[j] to 1 only if a majority of
26.             end for                                       // the nodes suspect it to have died
27.             if temp > n/2 then F[k] ← 1 else F[k] ← 0     // otherwise reset to 0
28.         end for
29.         for (k=0, k<n, k++)
30.             temp← 0
31.             for (j=0, j<n, j++)                                   // Check if all the fault-free
32.                 if S[j, k] ||  F[j]=1 then temp← temp+1           // members have detected
33.             end for                                               // the failed node
34.             if temp=n then consensus_reached(j)
35.         end for
36.         Append S to message when gossip is sent
37.     end do
38. end for
```

Fig. 2. The consensus algorithm.

The second step of updating the node's own row of the suspect matrix occurs every T_{gossip} seconds and is shown in lines 17-21 and 36. The process of examining heartbeats in the gossip list to check for liveliness is represented by the *fail_check()* procedure. If a dead node is suspected, the corresponding element in ***S*** is set to one. False failure detections will be removed in this step if the heartbeat value for a suspected node has been updated.

Lines 22-28 show how the suspect matrix is checked for a majority of nodes suspecting a given failure. If this condition is found, the corresponding element in ***F*** is set to one. The majority check, which is done every T_{gossip} seconds, prevents false detections from affecting the correctness of the algorithm by ensuring that only the faulty processors are masked. However, the requirement of a majority vote implies that the algorithm operates under the assumption that half or more of the nodes in the system will not experience a failure during a single consensus cycle. Although it has been omitted from the pseudo-code for simplicity, this majority check and the following consensus check are also performed along with the receipt of a gossiped suspect matrix. These extra checks ensure that that the consensus state is detected at the earliest possible time.

Finally, lines 29-35 illustrate the operations to be performed to check if consensus has been reached. To check if consensus has completed for node *j*, the logical OR is performed between the k^{th} column of the suspect matrix and the fault vector. The operation masks all faulty nodes by setting the corresponding element in ***F***, yielding a vector of ones if and only if all non-faulty nodes suspect the same failure. Once consensus is reached, the system may continue operating normally based on the reduced system size, thus reducing the size of the fault vector and suspect matrix, as represented by the *consensus_reached()* procedure in the pseudo-code.

4.2 The Round-Robin (RR) Protocol

The basic gossip protocol is random in nature, meaning that at any given time a node cannot determine the source of the next gossip message it will receive. It is also possible that a node receives no gossip messages for a period long enough to cause false failure detections. The gossiping traffic can be made uniform by employing a more deterministic approach. In the RR protocol, gossiping takes place in definite rounds every T_{gossip} seconds. In any one round, each node will receive and send a single gossip message. The destination node for each gossip message is determined by the following equation where *r* is the current round number:

$$\text{Destination ID} = \text{Source ID} + r,\ 1 \leq r < n \tag{1}$$

where *n* is the number of nodes. Using this method, it would take *n*-1 rounds for each node to establish one-to-one communication with every other node for a system size of *n* nodes. As a result, *r* is implemented as a circular counter with the initial and final value of 1 and *n*-1 respectively. Fig. 3 shows the communication pattern for the five rounds necessary in a six-node system.

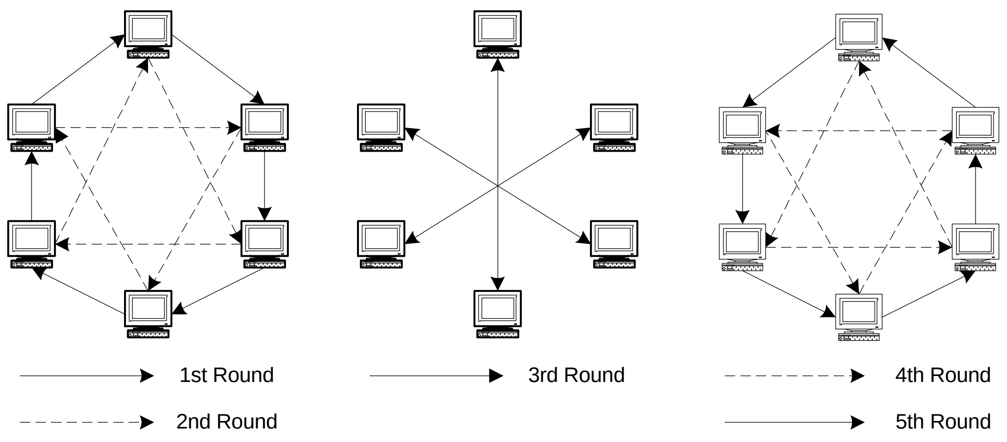

Fig. 3. Communication pattern in the round-robin protocol (n = 6).

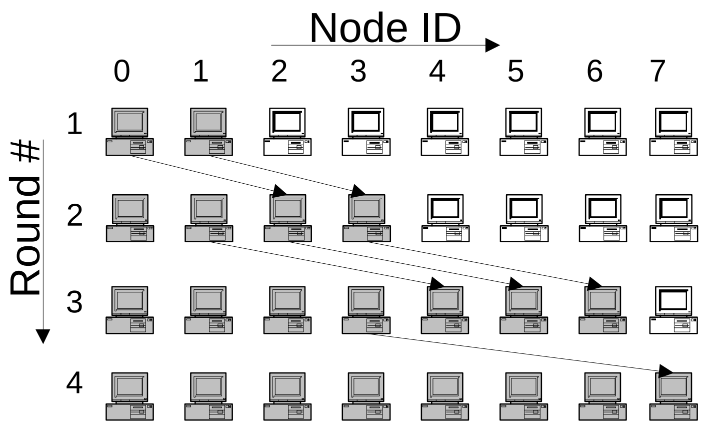

Fig. 4. Example of the gossip distribution in the RR protocol (n = 8).

Using a RR communication pattern guarantees that all nodes will receive a given node's updated heartbeat value within a bounded time. Fig. 4 illustrates how an updated heartbeat value for node 0 is distributed in an 8-node network. The nodes that have received node 0's message in a given round are shaded. In the first round, node 0 communicates its heartbeat to node 1. In the second round, node 0 communicates directly with node 2 while node 1 passes the updated heartbeat to node 3 and so on. As shown, it takes at least four rounds for all nodes to be aware of node 0's heartbeat. As a result, setting $T_{cleanup}$ to a value smaller than $4 \cdot T_{gossip}$ for a group of eight nodes will result in false failure detections and make consensus impossible. Similar measurements can be determined for various system sizes. Table 1 illustrates the relationship between round count and node count on round-robin systems, where three rounds can support up to seven nodes, four rounds can support up to eleven nodes, and so on.

Table 1. Maximum system size for a given number of rounds.

Number of Rounds	1	2	3	4	5	6	7	8	9	10
Max. Number of Nodes	2	4	7	11	16	22	29	37	46	56

A relation was derived to express this trend mathematically where α is the minimum number of rounds. This relation can be solved iteratively to determine the $T_{cleanup}$ value for a given system size. Generally stated, consensus for an n-node round-robin system is possible if and only if

$$T_{cleanup} \geq \alpha \cdot T_{gossip} \text{ , where } \frac{\alpha(\alpha-1)}{2}+1 \leq n \leq \frac{\alpha(\alpha+1)}{2}+1 \tag{2}$$

However, the value determined from the above expression is a worst-case calculation of the $T_{cleanup}$ parameter because the clocks of all the nodes are not tightly synchronized. While nodes gossip in a pre-determined order, all nodes do not gossip at exactly the same time within the period assigned to each round. Therefore, it is possible for one node to receive a gossip message from a second node before sending its own gossip message. If the data from the second node is incorporated into the first node's suspect matrix, it is possible for the first node to pass the second node's gossip data to another node in the same round. For example, in the first round of the RR protocol, node 1 may send its gossip data to node 2 *after* it receives node 0's gossip message. At this point, nodes 0, 1, and 2 will all have node 0's heartbeat value at the end of the first round. Therefore, it is possible for all nodes to receive node 0's message in less than four rounds, making α in Eq. 2 reflect merely an upper bound for the minimum $T_{cleanup}$ value.

Fig. 4 also shows that a measure of redundant communication takes place. After two rounds, nodes 1, 2 and 3 have received the gossiping information from node 0. In round three, node 0 sends another message to node 3 while a more optimal destination would have been node 4. Such redundant gossiping is eliminated in the binary round-robin protocol.

4.3 The Binary Round-Robin Protocol (BRR)

The BRR protocol attempts to make optimal use of the gossiping bandwidth by eliminating redundant gossiping. The redundancy inherent in the round-robin protocol is avoided by skipping the unnecessary steps. The algorithm is implemented by carrying out a minor modification of the round-robin protocol where *r* is the current round number:

$$\text{Destination ID} = \text{Source ID} + 2^{r-1},\ 1 \leq r < \log_2(n) \tag{3}$$

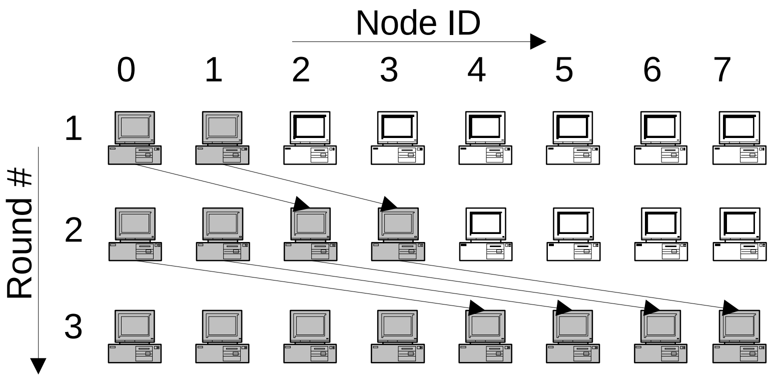

Fig. 5. Example of the gossip distribution in the BRR protocol (n = 8).

Using the same 8-node system example as before, the time taken for all nodes to receive the gossip of a single node was determined. The gossiping sequence is illustrated in Fig. 5. The figure clearly demonstrates that no redundant communication occurs when BRR is used. As a result, consensus in this case is possible as long as $T_{cleanup} \geq 3 \cdot T_{gossip}$. Once again, the expression is a worst-case calculation of the parameter because the clocks of all the nodes are not exactly synchronized. Generally stated, consensus for an *n*-node binary round-robin system is possible if and only if

$$T_{cleanup} \ \geq \ (\log_2 n) \cdot T_{gossip} \tag{4}$$

Table 2. Number of rounds required for a given system size.

Number of Nodes	2	4	8	16	32	64	128	256	512	1024
Rounds needed – RR	1	2	4	5	8	11	16	23	32	45
Rounds needed – BRR	1	2	3	4	5	6	7	8	9	10

The reduction in minimum allowable $T_{cleanup}$ will prove more useful as the number of nodes in the system increases because the number of rounds required in the BRR protocol is on the order of log(n) while the number of rounds required for the base round-robin protocol is worse. For example, Table 2 shows that a 1024-node system would require 10 rounds to fully disseminate gossip information using the BRR protocol whereas 45 rounds are required for RR. However, the BRR protocol in its present form works only for a system size that is a power of two. Modifications to the algorithm need to be investigated to implement the algorithm for an arbitrary number of nodes.

4.4 Round-Robin with Sequence Check (RRSC)

The deterministic nature of the round-robin protocol has advantages over the randomized approach of the basic protocol. Since every node knows how many gossips to expect and from whom to receive them, implementation of the protocol is greatly simplified. Moreover, the predefined order of gossiping allows a faulty node to be detected when the designated destination node does not receive a gossip message in a given round. Augmenting the RR protocol to leverage the communication sequence to generate failure detections yields a new gossip protocol, the Round-Robin with Sequence Check (RRSC) protocol. To implement RRSC, two modifications to the RR protocol are required:

1. A node is suspected to have failed if its gossip message is not received during a particular round.
2. A node previously marked as failed is added back to the gossip list if and only if that node is the source of a received gossip list.

The first change stems from the very nature of the sequence check protocol. To understand the relevance of the second modification, consider a 16-node system gossiping in a round-robin fashion. In the first, second and third rounds, node 1 would receive gossips from nodes 0, 15 and 14 respectively. If node 0 dies in the first round it will fail to gossip to node 1. In the second round when node 1 receives a gossip from node 15, the sequence check will reveal that node 0 had failed to send a message in the previous round. Therefore, node 0 is suspected to have died and is removed from node 1's list. In the third round, node 14, who does not suspect node 0 to have died, gossips its list to node 1. When the two lists are merged, if modification two is not used, node 0 will be added back to the list of node 1 and the result of the sequence check will be lost. Only when node 0 is the source of the message can node 1 be sure about node 0's state.

4.5 Scalable Consensus

Scalability is essential to failure-detection and consensus services for large systems. The gossip list, which contains a list of all the nodes in the system along with their heartbeats, increases in size as O(n) with the number of nodes. The consensus algorithm presented in this paper involves the gossiping of a suspect matrix of size n x n. If the consensus algorithm is implemented in a flat, monolithic manner, the suspect matrix will subsequently increase O(n^2) as the system size increases. The suspect matrix will also have to be updated every T_{gossip} seconds and the operations performed on the large matrices could prove to be unwieldy and would require more processing power and message-passing bandwidth.

We propose a scalable approach to failure detection and consensus by designing the service in a hierarchical fashion. In particular, a layered form of failure detection with consensus is employed in which the gossiping mechanism is divided hierarchically according to the network topology. Each node maintains three lists: a 'liveliness' list of all the live nodes in the system, an intra-cluster gossip list including heartbeats and suspect matrices of nodes within the cluster, and a smaller inter-cluster gossip list which logs an entry and a heartbeat for each cluster in the system. Fig. 6 shows an example of a hierarchical system in which a two-layered protocol is implemented. At the lowest layer, nodes within a cluster communicate the intra-cluster gossip lists while at the top layer, the inter-cluster gossip list is exchanged to detect network partitions. When consensus is reached about a failed node within a cluster, a message indicating the failure detection is broadcast to the rest of the system and the corresponding node is removed from 'liveliness' list. While group communication methods of failure detection need multiple broadcasts to detect a failure and thus depend heavily on an efficient broadcast mechanism, this protocol only requires a best-effort broadcast latency as the broadcast is not part of the failure-detection process and is used only once for updating the state of the system.

A similar consensus algorithm is implemented at the cluster level for detecting network partitions. The nodes within a cluster take turns in a round-robin fashion to transmit the inter-cluster gossip to a random node outside the cluster. This sequence results in a roving master for each cluster as opposed to a fixed master that could pose a reliability and performance bottleneck. To maintain a uniform view of the inter-cluster gossip list within the cluster, the list is attached to each intra-cluster message that is sent. Therefore, the size of the intra-cluster gossip message increases marginally with the system size.

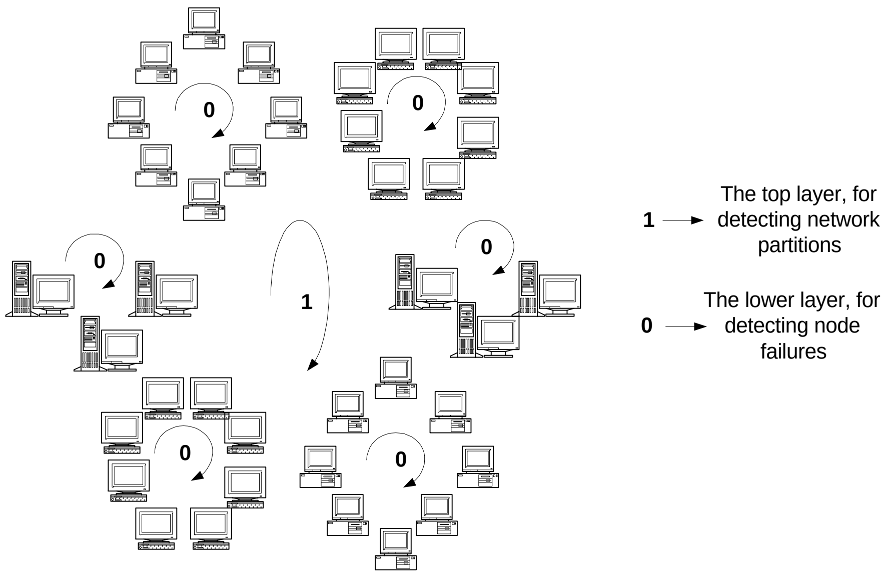
0
0
0
0
1
0
0
1 → The top layer, for detecting network partitions
0 → The lower layer, for detecting node failures

Fig. 6. Layered implementation of consensus in a large-scale system.

The separate layers may have different gossiping parameters, as network partitions are not as frequent as individual failures. Once a node fails, the time taken to reach consensus is largely independent of the system size and depends primarily on the size of the cluster in which the failure occurred, since consensus is achieved within the cluster and then propagated to the whole system.

5 SIMULATIVE ANALYSIS

In this section, the new gossip protocols coupled with the consensus algorithm described in the previous section are simulated and analyzed. The performance of the three round-robin protocols is compared to one another and to random gossiping. Finally, the scalability of the consensus scheme is examined with flat and layered designs.

5.1 Simulation Environment

The simulative experiments and analyses in this paper were conducted on a multiple-cluster, distributed computing system model designed using the Block Oriented Network Simulator (BONeS) [15] from Cadence Design Systems. BONeS is a flexible, event-driven simulation environment for designing and simulating complex networks using a CAD interface. The simulations in this paper leverage a high-fidelity BONeS model for Myrinet described in [7] that has been validated with both analytical and experimental analysis.

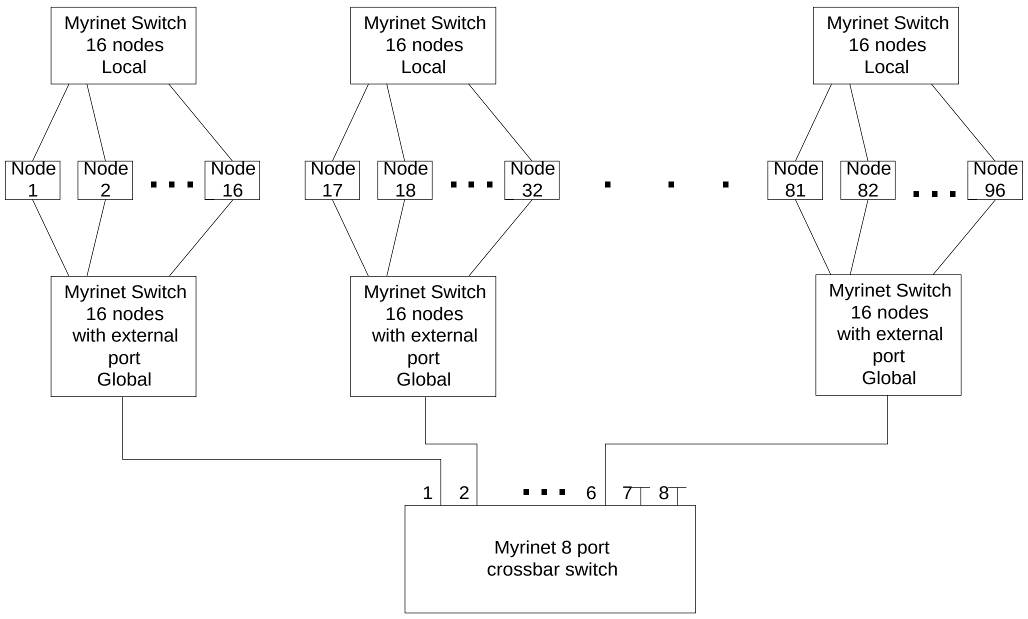

Fig. 7. Topology of the 96-node distributed system model.

The top-level design is based on a simulation model that mimics the structure of the system of clusters known as CPlant at Sandia National Labs, as described in [5]. Each cluster is comprised of at most 16 computing nodes connected via Myrinet, while a separate global Myrinet network is used to interconnect multiple clusters as shown in Fig. 7. Fig. 8 shows that each node contains two Myrinet interfaces, one interface that connects to the local network strictly for intra-cluster communications and one interface for communication with nodes in other clusters. The failure-detection service module shown in the figure generates gossips and processes received messages as specified by the consensus algorithm and the gossiping protocol being employed.

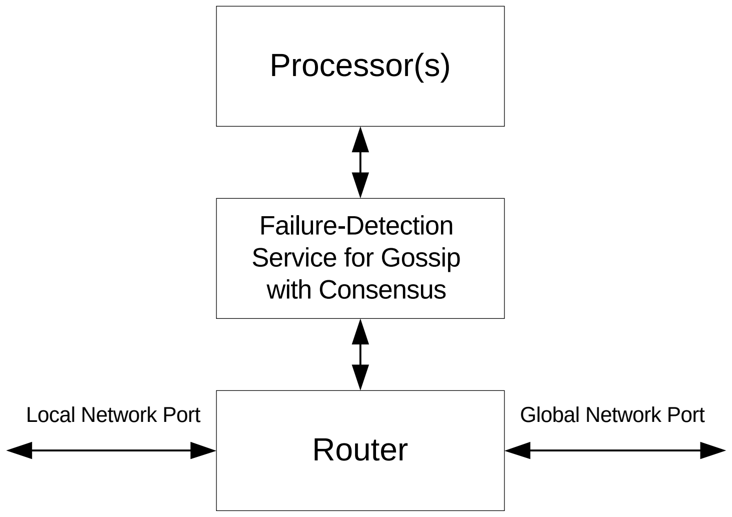

Fig. 8. Block diagram of a cluster node with the failure-detection service.

The processor module in each node consists of a Poisson traffic generator that produces messages with the destinations chosen at random. The rate of message generation was chosen to mimic the Parallel Direct Search (PDS) program [6], which is a parallel implementation of the direct search optimization method. The parallel program provided by Sandia National Laboratories is a controlling routine for the actual optimization and was found to generate application messages at the average rate of 4.8KB/s per node.

The gossip protocols were simulated on the distributed system model described above using functional-level, fault-injection techniques. The fault model used is based on fail-stop processor failures, and the number of processor failures is specified by a simulation parameter. At the start of the simulation the failed nodes are chosen at random and are made to operate in a fail-silent mode. Each simulation is initialized such that all the nodes are included in every gossip list and all the heartbeats are set to zero. The processor clocks are synchronized within a margin of error equal to T_{gossip} seconds. At each node the failure-detection service module examines the gossip list for failures and the suspect matrices for consensus. Once a node realizes that all the nodes have detected a failure, a message is broadcast to every other node to signal that consensus has been reached. When the broadcast message has reached all destinations, the simulation is terminated.

5.2 Protocol Simulations with Consensus

In order to compare the three round-robin protocols, simulations were conducted using a 16-node system with T_{gossip} set to 0.1 ms and a single failure injected. The time to reach consensus was then measured for each algorithm. The consensus time is defined as the time at which all nodes detect that consensus has been reached. Since the time taken to detect failures depends on the $T_{cleanup}$ parameter, the consensus time also varies with $T_{cleanup}$. A range of $T_{cleanup}$ values is simulated to find the minimum possible consensus time.

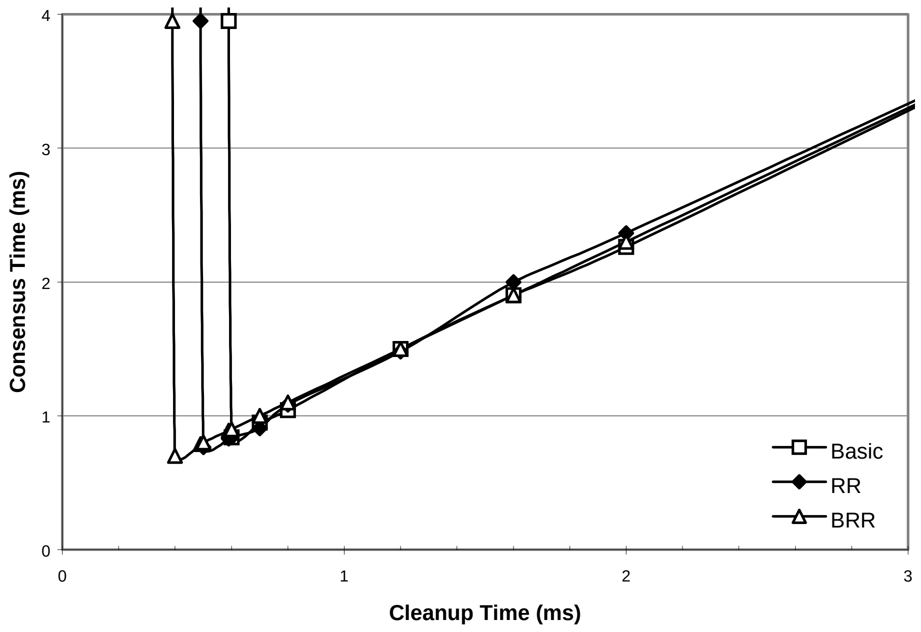

Fig. 9. Consensus time vs. cleanup time for a 16-node system (T_{gossip} = 0.1 ms).

Fig. 9 compares the consensus times of the basic, RR and BRR protocols. The figure shows that consensus is unattainable with the basic protocol for values of $T_{cleanup} < 6 \cdot T_{gossip}$ (i.e., $T_{cleanup} < 0.6$ ms in this case). A lower minimum $T_{cleanup}$ is possible with the RR protocol because of a more uniform gossiping mechanism that decreases the number of false detections. Consistent with the data in Table 1, RR allows a minimum $T_{cleanup}$ of $5 \cdot T_{gossip}$ (i.e., 0.5 ms) for the 16-node system. BRR allows for a minimum $T_{cleanup}$ of $4 \cdot T_{gossip}$ (i.e., 0.4 ms), which was specified in Eq. 4.

The simulation results in Fig. 9 present a worst-case scenario as the experiment was conducted on a tightly synchronized system with zero clock skew. Distributed systems will not have perfectly synchronized clocks, although one of the requirements of the gossiping protocol is that the synchronization error should not exceed T_{gossip} seconds. As explained in Subsection 4.2, clock skews may cause a non-deterministic improvement in the results. Simulation experiments conducted by incorporating a uniformly random amount of synchronization error ε (where $0 \leq \varepsilon \leq T_{gossip}$) yielded minimum $T_{cleanup}$ values that were less than the analytical upper bounds. For example, Fig. 10 shows the results of an experiment that compares the consensus times in a 16-node system for T_{gossip} = 0.5ms, achieving results similar to those in the previous experiment with the exception of a minimum $T_{cleanup}$ value with the BRR protocol of only $3 \cdot T_{gossip}$.

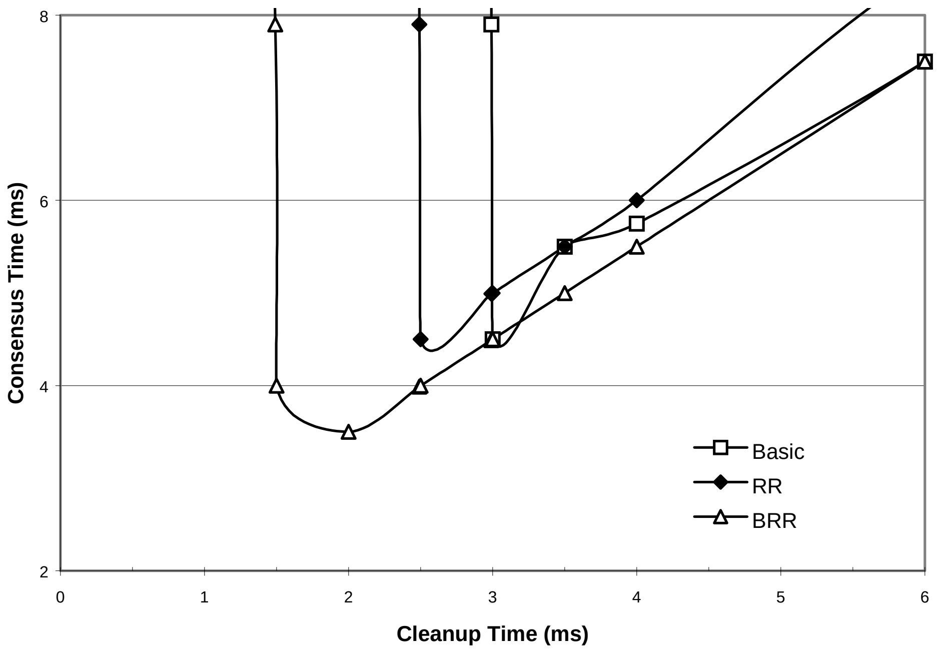

Fig. 10. Consensus time vs. cleanup time for a 16-node system with uniformly random clock skew (T_{gossip} = 0.5 ms).

In Fig. 11, the performance of the RRSC protocol is compared to the RR and the basic gossip protocol with respect to consensus time on a 16-node system, using a configuration identical to that used in Fig. 9. The data indicates that the minimum $T_{cleanup}$ for RRSC protocol is greater than that of the RR and the basic protocol. This limitation is due to the requirement in RRSC that false failure detections can only be resolved when a node receives a gossip directly from the falsely detected node. Therefore, false failure detections may not be resolved for low $T_{cleanup}$ values if the node does not participate in a direct communication with the falsely detected node. However, the results in the figure also demonstrate the advantage of performing sequence checks. The RRSC protocol requires (n-1) rounds of gossiping to reach consensus on the failed node purely through the use sequence checks, where n is the system size. Therefore, for values of $T_{cleanup} > 15 \cdot T_{gossip}$ (i.e., $T_{cleanup}$ > 1.5 ms in this case), the RRSC curve is virtually independent of $T_{cleanup}$. Independence from $T_{cleanup}$ is a useful property because it provides an upper bound on consensus time even if the system is not "tuned" to determine an optimal $T_{cleanup}$ value.

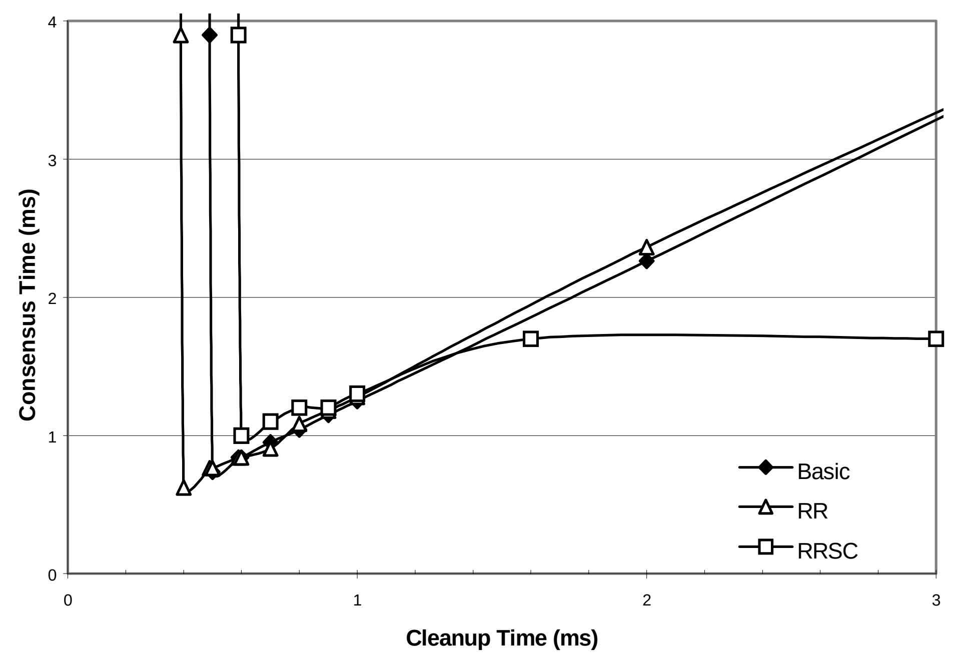

Fig. 11. Consensus time vs. cleanup time for a 16-node system (T_{gossip} = 0.1 ms).

Fig. 12 shows the time taken to reach consensus for simultaneous failures using the RRSC protocol. For values of $T_{cleanup} > 15 \cdot T_{gossip}$, the consensus time continues to be independent of $T_{cleanup}$. Though the minimum value for $T_{cleanup}$ does increase, consensus time increases only marginally for as many as six failures. The figure demonstrates the resilience of the RRSC protocol to simultaneous failures as well as its independence from the $T_{cleanup}$ parameter beyond a certain point.

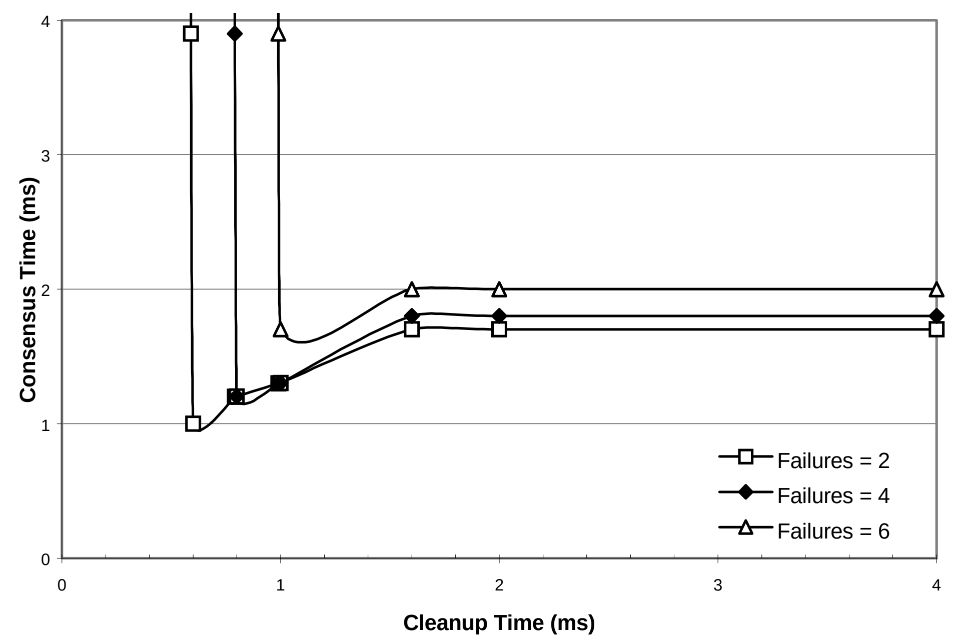

Fig. 12. Consensus time vs. cleanup time for multiple failures in a 16-node RRSC system (T_{gossip} = 0.1 ms).

5.3 Scalability Simulations

The consensus algorithm was also tested for its scalability with larger-sized systems. The consensus algorithm described thus far employs a flat or single-layered design in which consensus needs to be achieved among all nodes in the system in a global fashion. Fig. 13 shows the time taken to reach consensus with the basic protocol for system sizes of up to 80 nodes. The size of the suspect matrices increases as $O(n^2)$ with the system size resulting in larger communication delays and exponential increases in the consensus times. As a result, simulation times were prohibitive for system sizes greater than 80 nodes. However, using this trend, a conservative linear extrapolation for 96 nodes is shown.

The shape of the consensus time plot in Fig. 13 evidences the poor scalability of a flat implementation of the consensus algorithm. The consensus time for system sizes less than 48 nodes is relatively constant, but for system sizes approaching hundreds of nodes or more, the consensus times can be expected to increase rapidly. Meanwhile, as the figure demonstrates, the impact of an increase in the $T_{cleanup}$ parameter is simply a shift upward in the consensus curve by an amount proportional to the increase in the parameter.

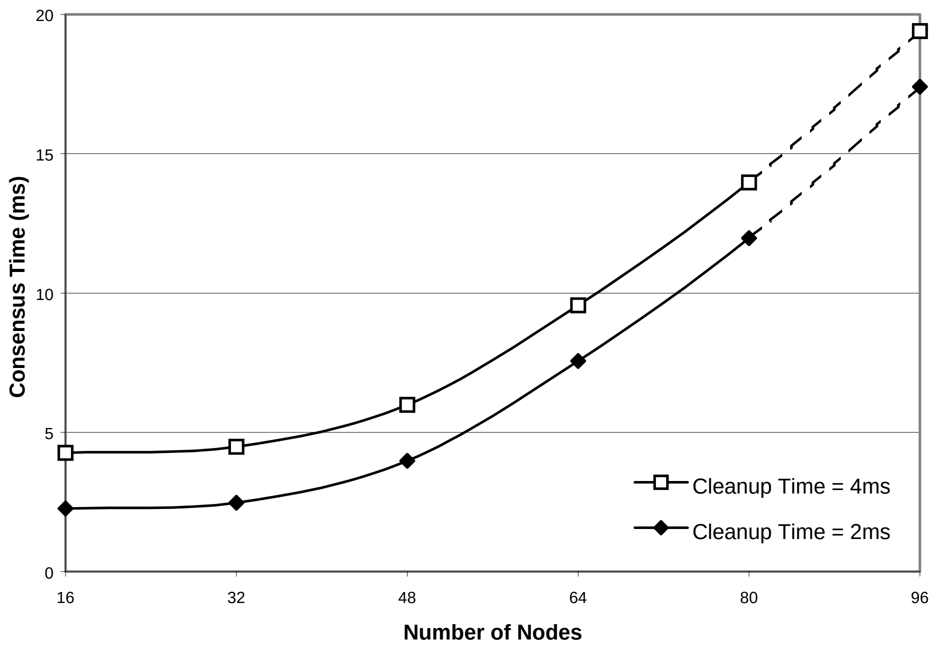

Fig. 13. Consensus time vs. system size for a flat implementation of the basic protocol (T_{gossip} = 0.1 ms).

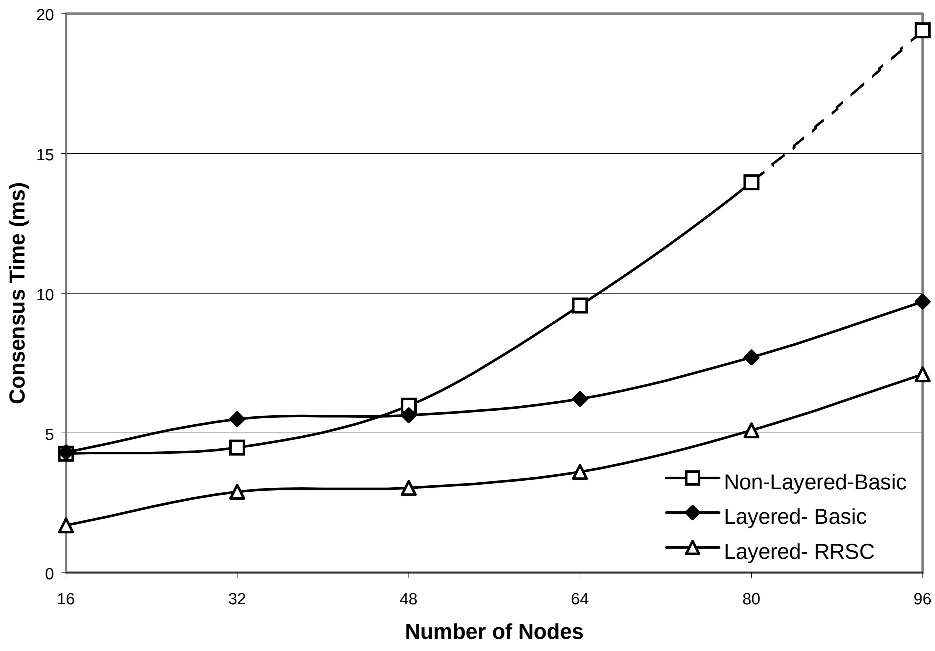

Fig. 14. Consensus time vs. system size for a layered implementation (T_{gossip} = 0.1 ms, $T_{cleanup}$ = 4 ms).

Fig. 14 demonstrates how a layered approach yields better scalability. In this experiment, the nodes are arranged into a varying number of 16-node clusters. For the 16-node test, there is one 16-node cluster while the 96-node test contains 6 such clusters. The plots compare consensus times between a flat implementation and a layered one for T_{gossip} = 0.1 ms and $T_{cleanup}$ = 4 ms. In the layered protocol, the top layer implements a basic gossiping protocol with either a basic or RRSC protocol in the lower layer. As the number of nodes increases beyond 48, the layered approaches scale linearly while the flat approach shows exponential growth. With the layered approach, as the system size increases, the consensus time increases only by the time taken to disseminate the information. Moreover, the bandwidth consumed is significantly reduced since the size of the message only depends upon the size of the cluster and not the size of the system. These advantages are evidenced in Fig. 14, which demonstrates the superior performance and scalability of a layered implementation of the consensus algorithm.

6 CONCLUSIONS

In this paper, several new gossip-style, failure-detection protocols are presented along with a novel algorithm for implementing consensus with them in either a flat or layered fashion. Several performance experiments are conducted to study the characteristics of these gossip and consensus techniques in terms of consensus time, cleanup time, and scalability.

Simulative studies indicate that the proposed gossip protocols provide the means for implementing an efficient failure-detection service with consensus for large-scale clusters. When compared to the basic randomized protocol, the round-robin protocol causes fewer false failure 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. Another benefit is that, for a desired value of $T_{cleanup}$, the more efficient gossip algorithms can use a larger value of T_{gossip}, thereby reducing the network overhead imposed by the gossiping. The binary round-robin protocol further optimizes utilization of gossiping bandwidth by eliminating redundant gossiping. For example, in general on a 16-node system, the BRR, RR and basic protocols yield minimum $T_{cleanup}$ values of $4 \cdot T_{gossip}$, $5 \cdot T_{gossip}$ and $6 \cdot T_{gossip}$ respectively. The round-robin protocol with sequence check is a resilient protocol that yields low failure-detection times without the need for configuring and tuning the $T_{cleanup}$ parameter.

Event-driven simulation studies on the basic gossiping protocols with consensus reveal the poor scalability of a flat implementation of the consensus algorithm. A layered approach to consensus is explored by designing the service in a hierarchical fashion according to the underlying network topology. The scalability of the algorithm improves because failure detection is divided into three logical stages: detection, consensus, and dissemination. The first two stages are independent of the system size because they involve only the nodes that are in close proximity to the failed node. In the analysis presented, the consensus time is the sum of the time for reaching consensus among the nodes in the cluster and for disseminating the information to the other clusters in the system. A layered design yields consensus times that scale linearly because increases in system size will only result in an increase in dissemination time, which can be minimized by using efficient broadcast schemes. The improved performance of the new protocols allows for the construction of a scalable failure-detection mechanism that yields efficient detection times with minimal processing and communication overhead. However, the performance improvements with the layered approach may be offset by a loss of resilience in the system, since the consensus protocol was not designed to support recovery if half or more nodes simultaneously fail within a single cluster. Therefore, the size of the clusters in a layered implementation of consensus, and the potential for common-mode failures within each cluster, become important considerations in the configuration of the system.

Many directions exist for future research on gossip protocols with distributed consensus. In terms of the design of the algorithms and protocols, further research is needed to find the optimal combination of techniques for basic, hierarchical, round-robin, sequence check, and other potential forms of gossiping with consensus designed to be implemented across two or more layers in a distributed system. For instance, in the design of layered systems, the tradeoffs between resilience and performance in terms of cluster size need to be investigated. Moreover, by incorporating compression techniques that reduce the size of the gossip messages, it may be possible to achieve the performance of a layered design with the resilience of a flat design. In addition to distributed consensus algorithms for dynamic detection of failed nodes in a system, consensus on the insertion or restoration of nodes is another direction of future research. Finally, experimental studies on large-scale clusters are also needed to implement these techniques on systems with heterogeneity in platform, network and operating system. The goal with such studies would be to evaluate critical performance characteristics in terms of such issues as failure recovery, task scheduling, operating system overhead, protocol stack overhead, network overhead, network utilization, CPU utilization, memory utilization, etc.

ACKNOWLEDGEMENTS

This research was supported in part by Sandia National Laboratories.

REFERENCES

[1] Aspnes, J. and Herlihy, M. Fast Randomized Consensus Using Shared Memory. *Journal of Algorithms.* Vol 11, No 4, 441-461, 1990.

[2] Babaoglu, O. On the Reliability of Consensus-based Fault-Tolerant Distributed Computing Systems. *ACM Transactions on Computer Systems.* Vol 5, No 3, 394-416, 1987.

[3] Barborak, M., Dahbura, A. and Malek, M. The Consensus problem in Fault-Tolerant Computing. *ACM Computing Surveys*. Vol 25, No 2, 171-220, 1993.

[4] Boden, N., Cohen, D., Felderman, R., Kulawik, A, Seitz, C., Seizovic, J. and Su, W. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*. Vol 15, No 1 , 26-36, 1995.

[5] Burns, M., George, A. and Wallace, B. Simulative Performance Analysis of Gossip Failure Detection for Scalable Distributed Systems. *Cluster Computing*. Vol 2, No 3, 1999.

[6] Dennis, J. and Torczon, V. Direct Search Methods on Parallel Machines. *SIAM Journal on Optimization.* Vol 1, No 4, 448-474, 1991.

[7] George, A. and VanLoon, R. High-Fidelity Modeling and Simulation of Myrinet System Area Networks. *International Journal of Modelling and Simulation* (to appear).

[8] Kuhl, J. and Reddy, S. Fault-Diagnosis in Fully-Distributed Systems. *Proceedings of the 11th International IEEE Symposium on Fault-Tolerant Computing*. 100-105, 1981.

[9] Kieckhafer, R. and Azadmanesh, M. Reaching Approximate Agreement with Mixed Mode Faults. *IEEE Transactions on Parallel and Distributed Systems.* Vol 5, No 1, 53-63, 1994.

[10] Nair, R. Diagnosis, Self-Diagnosis and Roving Diagnosis. *Dept. of Computer Science Rep. R-823*. University of Illinois at Urbana-Champaign, 1978.

[11] Preparata, F., Metz, G. and Chien, R. On the Connection Assignment Problem of Diagnosable Systems. *IEEE Transactions on Electronic Computers*. Vol 16, No 6, 1967.

[12] Taylor, K. and Golding, R. Group Membership in the Epidemic Style. *Dept. of Computer Science Rep. UCSC-CRL-92-1*. University of California at Santa Cruz, 1992.

[13] Tsuchiya, T., Yamaguchi, M. and Kikuno, T. Minimizing the Maximum Delay for Reaching Consensus in Quorum-based Mutual Exclusion Schemes. *IEEE Transactions on Parallel and Distributed Systems*. Vol 10, No 4, 337-345, 1999.

[14] Turek, J. and Shasha, D. The Many Faces of Consensus in Distributed Systems. *IEEE Computer*. Vol 25, No 6, 8-17, 1992.

[15] Shanmugan, S., Frost, S. and LaRue, W. A Block-Oriented Network Simulator (BONeS). *Simulation*. Vol 58, No 2, 83-94, 1992.

[16] Shostak, S. and Baker, B. Gossips and Telephones. *Discrete Mathematics*. Vol 2, No 3, 191-193, 1972.

[17] Van Renesse, R., Minsky, R. and Hayden, M. A Gossip-Style Failure Detection Service. *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing Middleware'98.* September 15-18, 1998.

[18] Yogen, D. and Oppen, D. The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment. *ACM Transactions on Office Information Systems*. Vol 1, No 3, 230-253, 1983.