An Efficient Topology-Adaptive Membership Protocol for Large-Scale Cluster-Based Services

Jingyu Zhou[§†] Lingkun Chu[§] Tao Yang[§†]

[§] Ask Jeeves Inc., Piscataway, NJ 08854
[†] University of California at Santa Barbara

Abstract

A highly available large-scale service cluster often requires the system to discover new nodes and identify failed nodes quickly in order to handle a high volume of traffic. Determining node membership promptly in such an environment is critical to location-transparent service invocation, load balancing, and failure shielding. In this paper, we present a topology-adaptive hierarchical membership service which dynamically divides the entire cluster into membership groups based on the network topology among nodes so that the liveness of a node within each group is published to others in a highly efficient manner. The proposed approach has been compared with two alternatives: an all-to-all multicast approach and a gossip based approach. The results show that the proposed approach is scalable and effective in terms of high membership accuracy, short view convergence time, and low communication cost.

1. Introduction

Many Internet services, such as AOL [1], Ask Jeeves [2], Google [13], MSN [20], and Yahoo [32], are hosted in large-scale clusters with thousands of machines. These services are typically organized into a multi-tiered architecture, with data partitioning and replication at each tier for high availability. In this way, frequent component and network failures due to hardware faults, software bugs and operational errors can be masked [21, 23, 26].

Membership service is an essential component of these Internet services. The goal of the membership service is to maintain a yellow page directory of all cluster nodes, including both the aliveness and service information. For a service consumer node, the aliveness information is often used to avoid sending requests to a non-functioning node, and the service information is used to make well-informed decision, load balancing for instance [10, 27]. The requirements for membership service for such applications are that 1) Every node needs to receive updated membership information within a delay as short as possible when there is a change. 2) The number of messages to propagate in the protocol needs to be minimized. 3) Tolerance of faults in nodes and networks needs to be considered for high availability.

The architecture of a membership service can be centralized, where a stand-alone server provides membership information to all the service nodes (e.g., Google FS [12]), or distributed, where every consumer maintains its own yellow page directory (e.g., Neptune [28]). Though easier to implement, the centralized approach is not scalable and the central server is a single point of failure. Furthermore, it introduces an additional delay during a service invocation for contacting the membership server first to look up service nodes.

In this paper, we propose an efficient, scalable, topology-adaptive, distributed membership protocol for large-scale cluster-based services. Nodes from a subnet form a group and leaders are elected to form an additional hierarchy. The protocol is topology-aware in the sense that cluster nodes are automatically divided into subgroups based on the physical network topology and a propagation hierarchy is formed adaptively on such a topology. Such a protocol reduces cross-network traffic and facilitates fault isolation when there is a failure in the network. The above scheme is more efficient and scalable than the naive broadcast-based approach and a gossip-style approach. Our evaluation confirms that this approach has high membership accuracy, short view convergence time, and low communication costs.

The rest of the paper is organized as follows. Section 2 describes the functionality and requirements of a membership service in cluster-based environments. Section 3 describes our topology-aware protocol. Section 4 analyzes the scalability of our approach with a comparison with two alternatives. Section 5 presents the details of the implementation and our evaluation in a cluster with hundreds of nodes. Section 6 discusses related work. Finally, Section 7 recaps our findings and concludes the paper.

2. Problem Definitions and Approaches

A membership service maintains the directory of all available service nodes, the types of their services, and other application-specific status information. This information is made available to all nodes. A membership service needs to detect a change quickly when a node enters or leaves a cluster or there is a network failure. With node membership information, when a node seeks a service from other nodes, it can avoid unavailable replicas that are listed providing such a service and select the best replica with the minimal workload. The consideration of networking infrastructure is important in designing an efficient membership protocol because the membership information is propagated through such a network. Figure 1 demonstrates a sample layout of an Internet service on clusters located in multiple hosting centers. Each center contains a large cluster of computer nodes which provides certain services to network users. A service may be available from multiple hosting centers or a service may be provided at one center and accessed from another center. Since typically network service applications localize communication and computation within a data center, this paper focuses on the design of an efficient membership protocol in a large-scale cluster within a data center. We have also extended our protocol to support membership information exchange among centers. Due to space constraint, its details are left to a technical report.

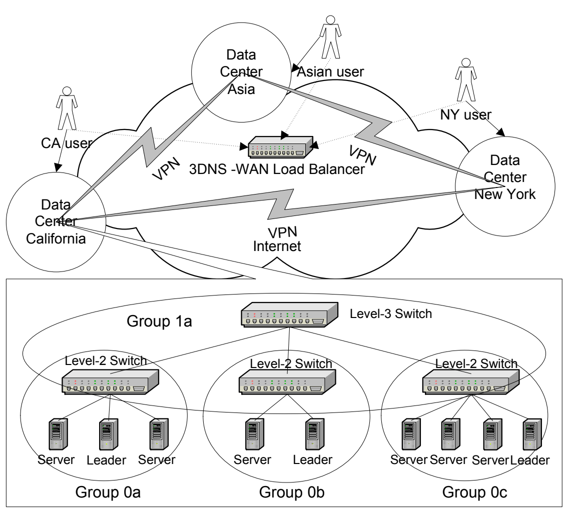

Figure 1: A sample cluster-based Internet service.

It is important that a membership service detects a node failure and notifies appropriate nodes as fast as possible (e.g. within few seconds for a large cluster). Figure 2 demonstrates the importance of fast notification with an example in an online document retrieval service we have investigated recently [6]. In this case, a server node becomes unavailable at time 7 and the failure is detected at time 12. Replication of this failed component does not help within the short period of time from time 7 to 12 because new requests are still sent to this failed node. As a result, too many requests are accumulated waiting and all system resources are exhausted. When all these nodes become unresponsive, the entire auction service becomes unavailable from time 10 to 13.

Another important consideration is to minimize communication needed in the membership protocol and tolerate faults caused by unexpected switch failure or network partitions. As a large-scale cluster typically uses level 2 and level 3 switches, exploiting the network topology can propagate information following a hierarchy and localize protocol traffic within physical switches which in turn minimize impact of switch failures.

Finally, the membership service should also quickly adapt to changes in the network topology because the topology can often change intentionally or unintentionally when there is a service expansion, a network device upgrade, or even an operational error.

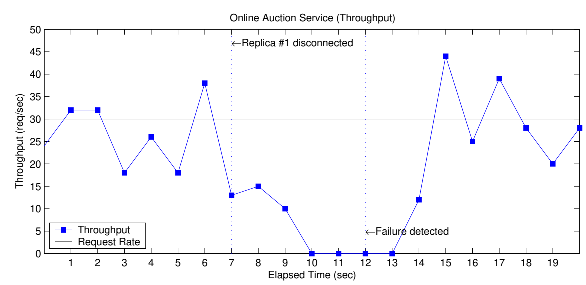

Figure 2: Service availability during the failure detection time.

One straightforward approach of a membership service is to let every node periodically send its heartbeats to other nodes and meanwhile collect heartbeats from other nodes [28]. This is an all-to-all approach. Each heartbeat packet contains service information and current load status of a node. Every node builds its own membership directory based on these heartbeat packets. This is a fully distributed approach in the sense that every node maintains its membership directory independently. This scheme works fairly well for a cluster of a small or medium size and it can tolerate switch failure and unexpected network partitions. But it is not scalable for a large cluster with thousands of nodes for which communication and computation overhead in maintaining a local yellow page directory can be significant[1].

1 It is possible to modify this algorithm with a lazy broadcast approach if only certain information is needed, such as a detected failure

We performed an experiment that measures the membership service overhead imposed on a Linux machine with dual 1.4 GHz P-III processors when varying the number of heartbeat packets received by this machine. The result of this experiment is shown in Figure 3. If each node sends a 1024-byte heartbeat packet per second, the heartbeat packets can consume 4MB/s bandwidth for each node in a cluster with 4000 nodes, which is 32% of the raw bandwidth of a Fast Ethernet link from this node. It should be noted that Gigabit cards for PC machines and a Gigabit switch with 24 Gigabit ports or less are cheap in today's market, however a high-end single Gigabit Ethernet switch with a large number of Gigabit ports (e.g. > 40) is still very expensive. As a result, many large-scale clusters with hundreds or thousands of machines in a production environment use Gigabit switches with fast Ethernet ports. It is possible to link a few Gigabit switches of 24 Gigabit ports, and then communication bandwidth among these switches is limited within few Gigabits and the overall all-to-all communication volume will easily surpass such a limit.

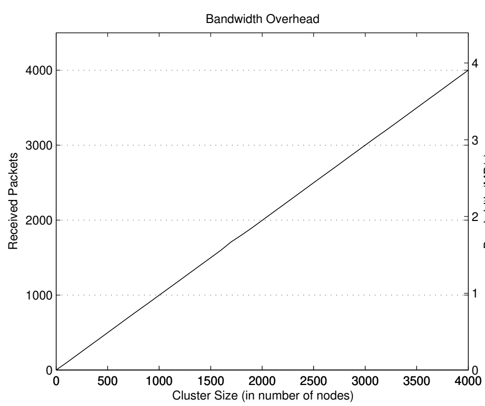

Figure 3: Communication overhead of the all-to-all approach as the cluster size increases.

An alternative approach used in wide-area network applications is gossip style membership service [24]. In a gossip style approach, each node randomly selects a set of neighbor nodes and sends them its current membership directory. The recipient nodes then update their own directories based on the new information. Given a rate of error tolerance, this approach can control the amount of heartbeat traffic by selecting an appropriate number of neighbor nodes. This capability is essential in wide-area services where bandwidth is limited, network latency is high and fast multicast is usually unavailable. When it comes to low latency and high bandwidth system area networks, a gossip style membership service has the problem of slow convergence time and high communication overhead for maintaining a complete view of the service cluster. Furthermore, its probabilistic property does not guarantee 100% accuracy, which can be unacceptable for some high reliable services.

Our intention is to pursue a hierarchical approach with a tree structure for communicating membership information in a large-scale cluster so communication cost can be reduced while change detection time is still competitive. Tree communication has been studied in the previous work in various different contexts and the challenge is to devise a scheme which is topology-aware and adaptive so that network communication is localized and minimized while network failure can be effectively tolerated.

(namely, broadcast only when there is a failure). Then such a scheme may not be general for providing a periodic update of other service status required by applications.

3. Topology-Adaptive Hierarchical Membership Service

For machines in a large-scale cluster, we form a hierarchical tree among nodes. Note that the underlying network topology does not necessarily need to be a tree. However, each node is assumed to join the membership group through only one network interface, which is generally true in practice. In general, an internal tree node passes the membership information from its parent to children nodes, and also it collects the information from its children and propagates to its parent. The membership information from one child of a node will be propagated to other children of this node.

In our protocol, an internal tree node and its children form a communication group. Multicast is used within the group for communication. A hierarchical tree is created based on the network topology of level-3 switches by exploiting the Time-To-Live (TTL) field in an IP packet header. The original purpose of TTL is to prevent packets falling into infinite routing loops. When an IP packet passes a router, the router decreases the TTL of this packet by one and forwards this packet to the right subnet. When the count reaches zero, the packet will be discarded. We exploit this feature to limit the scope of multicast packets within each communication group.

3.1. Topology-adaptive Group Formation for Node Joining

The key idea of our topology adaptive strategy is to form a number of small multicast groups among cluster nodes using the topology information. The overlapping of these multicast groups forms a hierarchical structure among nodes. The multicast within each group is highly efficient and the scalability of the membership protocol is achieved by dividing nodes into small groups. In this scheme, each node joins

```
join(local_addr)
{
    ttl = 1;
    channel = BASE_MULTICAST_CHANNEL;

    while (ttl < MAX_TTL) {
        join_group(channel, ttl);

        if (channel has a leader) {
            bootstrap with the leader;
            break;
        } else {
            elect_leader(channel);
            /*if there is only one node in
               this group, this node is the leader*/
        }
        if (is_leader(local_addr)) {
            ttl =ttl +1
            channel = next_channel(channel);
            continue;
        }
        break;
    }
}
```

Figure 4: Pseudo-code for a node to join a communication group.

a multicast group with the same TTL value and each multicast group has a communication channel (following the UDP protocol). As level 3 switches separate node multicast communication with different TTL value, a topology-aware hierarchy can be formed, which is also adaptive to any topology change later on. Figure 4 illustrates the group joining procedure performed by each node at startup time.

Initially when a node joins in, its TTL value is set to one and it uses the base channel of the membership protocol. By listening to the selected multicast channel, the node can find if there is a leader for the group. If there is a leader, the node will use a bootstrap protocol to quickly build its local yellow-page directory. Otherwise, an election process is performed. If this newly-joined node is a leader in this multicast group, it increases its TTL value and joins the membership channel at a higher level. This process continues until the maximum TTL count is reached, which is the largest possible hop count according to its network topology of the cluster.

The bootstrap protocol allows a newly joined node to quickly build its local yellow-page directory. After knowing the leader, the node contacts the leader to retrieve the membership information that the leader knows. Meanwhile, the leader of this communication group also queries this newly joined node for its membership information in case that the new node is a leader for another group in a lower level. When the new information is obtained, the group leader of this new node propagates the information further to all group members using an update propagation protocol, which will be discussed in Section 3.3.

Group election determines a leader for a group using the bully algorithm [5]. Each node is assigned a unique ID (e.g., IP address). The node with the lowest ID becomes the group leader. If there is already a group leader, a node will not participate the leader election. To reduce the chance of re-election due to failure of a leader, each group maintains a group leader and a backup leader. The backup leader is randomly chosen by the group leader and it will take over the leadership if the primary leader fails. This allows quick recovery if only the primary leader fails. When both the primary and the backup leader fail, the election algorithm is performed to select a new leader, which will designate a backup leader thereafter.

It should be noted that if a group only has one member (i.e. a first joined node), then this node is the default leader. There could be a number of multicast groups with one member, especially when the joining process reaches a high TTL value. We still keep such groups for topology adaptivity because some new nodes may join in the future. Since the maximum TTL value is normally small in a large cluster, there should not be many such groups. The communication cost is negligible for groups with only one member, because the corresponding router multicast tree is very small.

3.2. Properties and Examples

The above group formation process creates a hierarchical tree where leaves are cluster nodes and internal tree nodes are leaders elected. We define the level of a group as the TTL value of the group minus one, and we can prove that a hierarchical tree derived based on the topology has following properties:

- All alive nodes in the cluster will be eventually included in the hierarchical tree if there is no network partition. With network partitions, forests will be formed and every node will be in one of trees. That means that the status change of a node will be propagated to all nodes connected in the same tree.
- If a node is present in a group of certain level, it is aware of the group leader when the group is in the steady state. Therefore, it can inform its group leader when a change is detected, i.e. messages can be propagated upwards.
- If a node is present at level i, it must join as leaders in lower level groups, 0, 1, ..., $i-1$. This means a change message at level i will be propagated to lower levels, i.e. messages can be propagated downwards.

We illustrate the node joining protocol with two examples. The first example is based on California data center of Figure 1 in Section 2. In total, four multicast groups have been formed to build a hierarchical membership propagation tree with the maximum TTL value as 2. Groups $0a$, $0b$, and $0c$ are formed with TTL value one. With TTL value one, packets from one member of these three groups cannot pass the level-3 switch to reach another group. Each of the three groups elects a leader, which joins Group $1a$ with TTL value two.

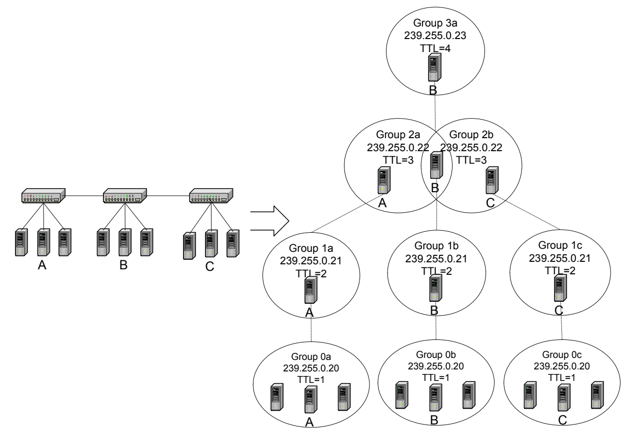

Figure 5: The left and the right of the figure show physical network layout and membership groups respectively.

The second example is illustrated in Figure 5 with nine multicast groups formed in this case for hierarchical communication. Nodes A, B and C are leaders for groups $0a, 0b$, and $0c$ respectively. In the next level, node A forms its own group with TTL=2. The reason is that node A cannot reach node B with two hops. Then nodes A and B form a group called 2a with TTL=3 because they can reach each other with 3 hops. Similarly node B and C form group 2b with TTL=3. We assume that B is selected as a leader in group 2a. In group 2b, B is also elected. Then B forms a group by itself with TTL=4.

3.3. Update Protocol for Cluster Change

Membership change should be made aware to the nodes in the entire cluster. In our scheme, a multicast group leader propagates such information promptly by notifying its members and its parent group.

We have discussed the node joining process in the above subsection. For detecting the departure of a node due to failure or an operation decision, we use the heart beating method. A node always continuously multicasts its availability (heartbeat messages) in each multicast group it resides. Since multicast heartbeat packets may get lost, a node is considered dead *only* when no heartbeat packet is received from the node after a pre-defined time period.

When a multicast group leader receives an update from its child group, it needs to further multicast the update information in its parent multicast group. Similarly when a leader receives an update message from its parent multicast group, it needs to multicast the new information to its child multicast group. In this way, an update of node status can be propagated to the entire cluster quickly. Figure 6 illustrates the propagation of an update message. Nodes B, E and H are the leaders of groups $0a, 0b$, and $0c$ respectively. Node E is the leader of group $1a$. Assume node C is dead in group $0a$, and node B detects this failure and removes C from its membership directory. Then node B also multicasts this update to its parent group $1a$ at Step 2. At step 3, node E forwards this information to all nodes in group $0b$ through multicast after it receives this update at group 1a. At step 4, all nodes at group $0b$ update its local membership directory. Concurrently, all nodes at group $0c$ update its membership directory also and exclude node C.

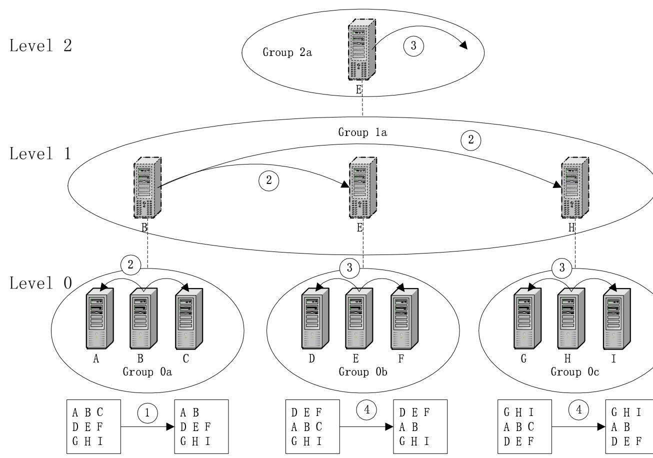

Figure 6: Propagation of an update message. The circled numbers show the propagation order of the update message.

We want to add two techniques in the design of the above update protocol.

- **Handling unexpected network partitions or switch failure with hierarchical timeout**. When a leader brings in new information on the group or subgroup it manages, other groups need to remember that this leader is in charge for the specific group. If a group leader is considered to be dead, then all nodes managed by this leader are considered dead tentatively by other groups, mainly for detecting a switch failure.

  For example in Figure 6, if node B is dead, it is possible that it is caused by an unexpected network partition or switch failure so that all nodes in group 0a can be non-accessible from groups $0b$ and $0c$. The other multicast groups in the system first assume that all nodes in group $0a$ are dead and purge them from the membership directory as a possible switch failure, and then let the new leader in group $0a$ re-announce their availability in the subsequent iteration.

Since it takes time to remove assumed dead nodes from the membership table of a node and then add them back, to minimize the impact of handling the failure of a leader, different timeout values are assigned for multicast groups at different levels. Higher level groups are assigned with larger timeout value, namely a longer heartbeat waiting period in determining if a node is dead or not. In this way, when a group leader fails, a new leader can be quickly selected to replace the old leader before the higher level group detects the failure. This can avoid the unnecessary purging of nodes from the available membership directory.

- **Delta information updating and recovery of lost messages.** When a leader updates new information to a multicast group, it only announces the changed portion to minimize communication size. Because UDP multicast packets can be lost during network transmission, to help detect a packet loss, each host assigns a sequence number for an update message. Thus the receiver can use the sequence number to detect lost updates. Since each update about a node departure or join has a very short message, we let an update message piggyback last three updates so that the receiver can tolerate up to three consecutive packet losses. If more than three consecutive packets are lost, the receiver will poll the sender to synchronize its membership directory.

4. Scalability Analysis

In this section, we present analytic results to compare the hierarchical approach with two alternatives: the all-to-all approach and the gossip approach in a cluster environment. We assume multicast is used to disseminate messages to a group of nodes, and unicast is used for one to one communication, such as gossip messages.

We use the following three metrics for a comparison:

- **Failure detection time** (T_{fail}) is the earliest time that a failure is detected by any of other nodes. Notice that when a failure is detected by one node, it may not be known to others.
- **View convergence time** is the length of an interval from the time of a status change to the time that all nodes have a consistent view of the change.
- **Communication cost** in maintaining a protocol. We measure it using communication bandwidth consumption requirement per second.

Since fast detection can require more communication volume to make sure a new event is detected promptly while we also prefer low communication cost, we use the metric BDP that combines bandwidth consumption and failure detection time. Assume the bandwidth consumption of a scheme is B bytes per second when the cluster status is stable, the metric is calculated as $BDP = B \times T_{fail}$, where T_{fail} is the detection time of a node failure. Protocols with lower BDP values are better, because they use less time to detect a failure with a fixed bandwidth.

4.1. Failure Detection Time and Communication Cost

Let m be the average size of a protocol message, n be the total number of nodes in a cluster, and B be the total bandwidth allowed. The failure detection time and BDP are calculated as follows.

- **All-to-all.** The multicast frequency is limited by $f = \frac{B}{m \times n^2}$ since each node receives heartbeats from all other nodes and sends out one heartbeat per multicast cycle. Thus each node consumes $m \times n$ bandwidth per cycle. If the protocol assumes a node is dead after not hearing p consecutive heartbeats from the node, the failure detection time is

$$T_{fail} = \frac{p}{f} = \frac{p \times m \times n^2}{B},$$

and

$$BDP = B \times T_{fail} = p \times m \times n^2 = O(n^2).$$

In practice, each node often fixes its multicast frequency, which is independent of the number of nodes. This makes the failure detection time as a constant and then the total amount of network bandwidth consumption B will become $O(n^2)$.

- **Gossip.** Each gossip message contains a local view of cluster membership. As each node accumulates the global view incrementally following a randomized manner, the size of the local view reaches $m \times n$ bytes eventually. Then bandwidth consumption is $O(mn^2)$. We do not count the broadcast message in the gossip scheme since it can be eliminated under optimization. The frequency of gossip can be calculated as $f = \frac{B}{m \times n^2}$. For this scheme, the previous research shows that the number of failure detection steps is $O(\log n)$ [24]. Thus we compute the failure detection time and BDP as:

$$T_{fail} = \frac{O(\log n)}{f} = \frac{O(\log n) \times m \times n^2}{B},$$

and

$$BDP = B \times T_{fail} = O(n^2 \log n).$$

If the gossip approach consumes $O(n^2)$ amount of bandwidth, then the failure detection time will be $O(\log n)$.

- **Hierarchical.** Assume the size of each multicast group with a varying TTL value is limited to a constant of k nodes, the height of the membership tree is limited by $\log_k n$. Adding up the number of groups at each level, we get the number of total groups

$$g = \frac{n}{k} + \frac{n}{k^2} + ... + \frac{n}{k^{\log_k n}} = \frac{n-1}{k-1}$$

The multicast frequency is $f = \frac{B}{(g \times m \times k^2)}$ since each group has k nodes which consume $m \times k^2$ bandwidth. If the protocol assumes a node is dead after not hearing p consecutive heartbeats from the node, the failure detection time is

$$T_{fail} = \frac{p}{f} = \frac{p \times g \times m \times k^2}{B},$$

and

$$BDP = B \times T_{fail} = p \times \frac{n-1}{k-1} \times m \times k^2 = O(n).$$

If each node fixes its multicast frequency as the all-to-all approach does, the failure detection time is a constant and the total amount of network bandwidth will become $O(n)$, which is more scalable than the other two methods.

4.2. View Convergence Time

View convergence time includes the failure detection time and the time to disseminate this information to all other nodes. Similarly, we can define a metric BCP which combines bandwidth consumption with convergence time, measure the effectiveness of the three approaches: $BCP = B \times T_{converge}$.

For the Gossip and the flat all-to-all scheme, the view convergence time is the same as their failure detection time since all nodes maintain their views independently. Thus, their BCP values are $O(n^2)$ and $O(n^2 \log n)$, respectively. For the hierarchical scheme, the view convergence time is the failure detection time plus the time to disseminate this information along the hierarchical tree whose height is $\log_k n$. An update message will first travel up to the root of the tree and propagate down to the bottom of the tree. Assume the network transmission time of an update message is λ, the whole propagation will take $2\lambda \log_k n$. Thus, the convergence time is

$$T_{converge} = T_{fail} + 2\lambda \log_k n,$$

and

$$BCP = B \times T_{converge} = O(n) + O(B \times \log_k n).$$

In the worst case, B can be $O(n)$, the hierarchical scheme has the best scalability in terms of BCP.

If we just look at the view convergence time assuming that high communication cost is allowed, the all-to-all method has a constant time while the gossip method has $O(\log n)$ delay and the hierarchical approach have $O(log_k n)$ delay. In practice, the convergence rate for the hierarchical method can be very fast because λ is very small and k can be chosen quite large. The gossip method can be slow in view convergence since updating randomly does not follow a deterministic notification path.

In summary, the all-to-all method has a short convergence rate and failure detection time, but it is not scalable in terms of communication cost. The communication scheme in gossip method is also not scalable and its failure detection time is slower than others. The hierarchical method can have a reasonable failure detection time and convergence rate with a scalable communication scheme.

5. Implementation and Evaluation

In this section, we first illustrate the implementation of the hierarchical membership service, which is then evaluated and compared with two alternative approaches. The main objective is to study the scalability of the hierarchical approach in terms of failure detection time, view convergence time and network overhead.

5.1. Implementation

We have implemented the membership service in the *Neptune* framework – programming and runtime support for building cluster-based Internet services [28]. However, it can be easily coupled into other clustering frameworks.

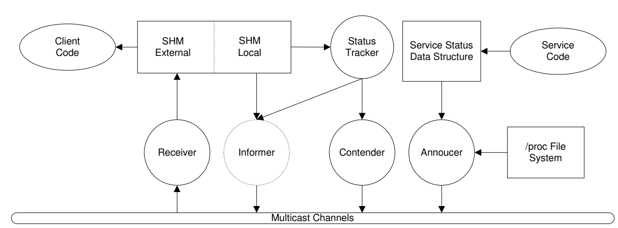

Figure 7: Implementation of the hierarchical membership (Circles and eclipses represent active entities, rectangles represent data structures and arrows show the information flow).

Figure 7 shows the components in our implementation and the related external entities. The components in circles are threads associated with specific tasks. The *Announcer* thread collects the machine information from Linux /proc file system and the service information from the IPC channel of the service. Then it publishes this information to the multicast channels the node has joined.

The *Receiver* thread subscribes to the multicast channels that the node has joined and updates the shared memory structure to reflect newly received information. The shared memory block is divided into two parts: (1) a local part that contains the information about all the directly connected nodes via the multicast channel. (2) an external part which contains information of external groups relayed by a group leader. The difference is that a node is responsible for detecting a failure in the local part while it depends on its group leader to tell the availability of an external node.

The *Status Tracker* thread periodically checks the entries in the shared memory block and purges any expired entries. When there is an expiration and the failed node is the leader of the local group, the Tracker will assume the backup leader as the new leader. If there is no backup leader, the Tracker wakes up the *Contender* thread to initiate an election process for a new group leader. If the node itself is a group leader, it needs to propagate a status change to higher level groups. This is done through the Informer thread, which propagates a change to other groups.

Besides relaying changes to other group members, the *Informer* thread of a group leader also listens to a well known UDP port. Thus, the Receiver thread on a newly joined node can poll Informer to get the entire yellow page. Furthermore, the Receiver is also responsible for detecting any loss of update packets. Each update packet contains last three status changes to improve the tolerance of the packet loss. If there is an unrecoverable loss, the Receiver will also poll the source node to get a complete image.

5.2. Experiment Settings

All the experimental evaluations were conducted on a rack-mounted Linux cluster with 100 dual 1.4 GHz Pentium III nodes. Each node runs RedHat Linux (kernel version 2.4.20). There are two Layer-3 switches with 100Mb links. One accommodates 50 nodes each. These two switches are connected by a Gigabit link.

For our hierarchical protocol, we manually designate multicast channels to emulate multiple networks. Each multicast channel hosts 20 nodes. Therefore, there are five networks for 100 nodes and these five networks form a second level network.

For Gossip scheme, mistake probability is set to 0.1%, which represents the bound that any node may make an erroneous failure detection. This is a relatively loose requirement for the Gossip scheme. As discussed before, each gossip message accounts for a number of network packets and the broadcast packets are not counted.

In the following experiments, we fix the multicast or gossip frequency as one packet per second for all three schemes. For the all-to-all scheme and the hierarchical scheme, we set the maximum packet losses as 5 before a node is considered as dead. We vary the number of nodes from 20 to 100 with the number of networks from 1 to 5. The average packet size carrying the membership information of each node is measured as 228 bytes for all three schemes.

5.3. Bandwidth Consumption

First, we compare the bandwidth consumption for three schemes in Figure 8. Bandwidth consumption is measured on each node by counting the incoming heartbeat packets. Then all numbers are added up to get the aggregated bandwidth consumption. When there are 20 nodes, all the schemes use the same amount of bandwidth. When the number of nodes grows, the hierarchical scheme has the least total bandwidth consumption and has close to linear growth. On the contrary, the bandwidth usage for both the all-to-all scheme and the gossip scheme grows quadratically with the number of nodes. These results are in line with our analysis results in Section 4, where we show that the average bandwidth consumption for each node remains constant in the hierarchical approach and grows linearly for the other two approaches. This suggests that the hierarchical approach is more scalable in terms of network bandwidth usage.

5.4. Failure Detection Time

Figure 9 shows the failure detection time for three schemes. During the experiments, a membership service daemon process on a node is killed to emulate the node failure. We can see from the figure that as the number of nodes grows, the hierarchal scheme and the all-to-all scheme have the same constant detection time which is around 5 seconds, the maximum number of packet losses times the multicast period. On the other hand, the detection time of the gossip scheme increases logarithmically along with the number of nodes. It also has the longest detection time when there are only 20 nodes. Both the hierarchical and the all-to-all schemes have shorter failure detection time than the gossip scheme. This experiment results are also in accordance with our analysis in Section 4.

5.5. View Convergence Time

Figure 10 compares the view convergence time. The hierarchical scheme has the similar view convergence time as the all-to-all scheme. This is because they have the same

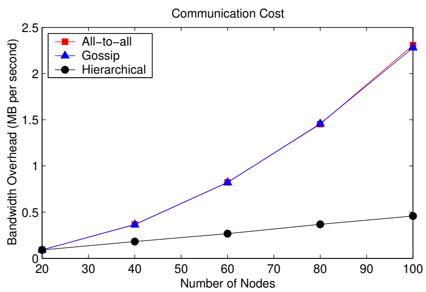

Figure 8: Bandwidth consumption

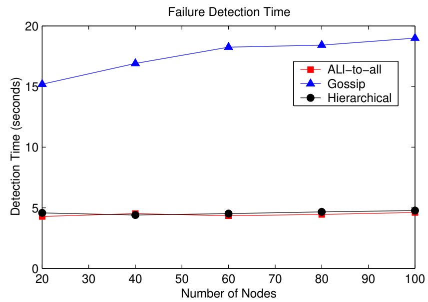

Figure 9: Failure detection time

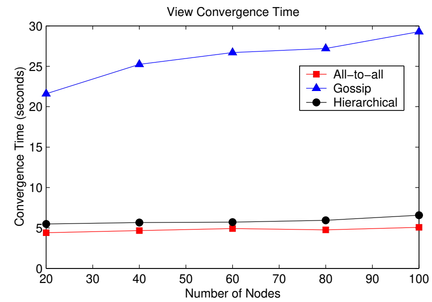

Figure 10: View convergence time

failure detection time and when a failure is detected, group leaders can quickly propagate this information to all nodes. Figure 10 also shows the view convergence time of the gossip scheme is the biggest among the three schemes and it grows along with the number of nodes. Again, the hierarchical and the all-to-all schemes perform better than the gossip scheme.

5.6. Discussion

From the above experiments, we can see that gossip scheme performs the worst. Using the same amount of network traffic, it has the longest detection and convergence time. When the number of nodes scales up, the gossip scheme increases bandwidth usage quadratically. The reason is that each gossip has to carry a host's local view of the group membership, while the other two protocols use much smaller message size. If the gossip protocol is organized into a hierarchical fashion [24, 11], the detection time and convergence time will be longer due to cross group gossips. However, it is shown that Gossip protocol is useful for large groups where each member only needs a partial view of group membership [11]. Gossip style approaches are also attractive when efficient multicast is not available, for example in wide-area applications. We focus on protocols, such as the all-to-all scheme and the hierarchical scheme, which allow each member to quickly manage a global view. The hierarchical scheme is better than the all-to-all scheme for its less bandwidth consumption and comparable performance. The experiment results closely match our analysis in Section 4. This validates our analysis and allows us to predict that our findings will remain valid for larger clusters.

6. Related Work

Previous research on membership or failure detection protocols for fault-tolerant distributed applications requires precise membership services to support other distributed protocols such as atomic broadcast protocols [4, 8, 22, 9]. Stok *et al.* [29] described a hierarchical membership protocol and their protocol requires all nodes have synchronized clocks so that the execution steps of the protocol can be synchronized. Our work is focused on network topology-aware techniques at the software application layer.

Gossip style membership services are different from heartbeat-based membership services in the aspect that they are based on probabilities [24, 16, 11]. These protocols are most attractive when full group membership is not required, especially in wide-area applications. For instance, SCAMP [11] is a hierarchical variation of gossip protocols where group members only have partial knowledge of the group. We focus on the cluster applications which require full membership knowledge.

There are similar ideas of hierarchically organizing group members in the research of overlay networks [3]. Most of work is focused on wide area network applications, where network latency is high and link bandwidth varies from node to node. Failure detection time and view convergence time are often not the primary goals.

Many high-availability systems [17, 14] also include membership services as an essential component. The Linux-HA project provides a heartbeat based membership service [18]. It enables a hot-standby feature that one machine can take over another machine's IP when it fails. But it only works in small scale. The study of [15] proposes a failure detector protocol for Grid environment. Our study aims at large-scale clusters with low latency and high throughput system area networks.

Cluster-based network services have been studied in [10, 30, 26, 28] and the membership service is part of infrastructure. Resource monitoring tools, such as Ganglia [25], the Network Weather Service [31], and the Monitoring and Discovery Service (MDS2) [19, 7] of Globus project, provide information of machine resource and network resource of large-scale clusters or computer grids. These projects do not emphasize fast failure detection time, convergence time, and topology awareness in details. It should also be noted that our membership service is general and can provide a periodical update of service status information for each node in addition to failure detection.

7. Concluding Remarks

The contribution of this work is a topology-adaptive hierarchical membership service for large-scale clusters. The key strategy is to form a number of small TTL based multicast groups with an update protocol to achieve fast hierarchical communication. It tolerates switch and node failure adaptively while localizing protocol communication following the network topology. Our evaluation shows the hierarchical membership service is scalable and efficient in large-scale clusters.

Acknowledgments

We thank the anonymous referees for their helpful comments on earlier drafts of this paper. This work was supported in part by Ask Jeeves and NSF grants CCF-0234346.

References

[1] America Online. `http://www.aol.com`.

[2] Ask Jeeves Search. `http://www.ask.com`.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. of ACM SIGCOMM'02*, Aug. 2002.

[4] T. D. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proc. of the 15th ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 322–330, New York, 1996.

[5] R. Chow and T. Johnson. *Distributed Operating Systems and Algorithms*. Addison-Wesley, 1997.

[6] L. Chu, K. Shen, H. Tang, T. Yang, and J. Zhou. Dependency Isolation for Thread-based Multi-tier Internet Services. In *Proc. of the IEEE INFOCOM*, Miami FL, Mar. 2005.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *IEEE International Symposium on High-Performanc e Distributed Computing (HPDC)*, Aug. 2001.

[8] C. Fetzer. Enforcing perfect failure detection. In *21st Proceedings of the International Conference on Distributed Computing Systems (ICDCS2001)*, Phoenix, AZ, 2001.

[9] C. Fetzer and F. Cristian. A highly available local leader election service. *Software Engineering*, 25(5):603–618, 1999.

[10] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *ACM SOSP*, Saint Malo, Oct. 1997.

[11] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2), February 2003.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *ACM SOSP*, 2003.

[13] Google Search. `http://www.google.com`.

[14] Cluster Infrastructure for Linux. `http://sourceforge.net/projects/ci-linux`.

[15] A. Jain and R. K. Shyamasundar. Failure Detection and Membership Management in Grid Environments. In *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 44–52, Pittsburgh, PA, Nov. 2004.

[16] D. Kempe, J. M. Kleinberg, and A. J. Demers. Spatial gossip and resource location protocols. In *ACM Symposium on Theory of Computing*, pages 163–172, 2001.

[17] High-Availability Linux Project. http://www.linux-ha.org.

[18] Linux Heartbeat. http://www.linux-ha.org/heartbeat.

[19] MDS2. `http://www.globus.org/mds`.

[20] MSN Groups Service. http://groups.msn.com.

[21] K. Nagaraja, X. Li, R. Bianchini, R. P. Martin, and T. D. Nguyen. Using fault injection and modeling to evaluate the performability of cluster-based services. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003.

[22] G. Neiger. A new look at membership services. In *Proceedings of the fifteenth Annual ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, 1996.

[23] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do Internet services fail, and what can be done about it? In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, Mar. 2003.

[24] R. V. Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proc. Middleware 98*, 1998.

[25] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler. Wide area cluster monitoring with ganglia. In *Proc. of the IEEE Cluster 2003 Conference*, Hong Kong, 2003.

[26] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-based Mail Service. In *Proc. of the 17th SOSP*, pages 1–15, 1999.

[27] K. Shen, T. Yang, and L. Chu. Cluster Load Balancing for Fine-grain Network Services. In *Proc. of International Parallel & Distributed Processing Symposium*, Apr. 2002.

[28] K. Shen, T. Yang, L. Chu, J. L. Holliday, D. A. Kuschner, and H. Zhu. Neptune: Scalable Replication Management and Programming Support for Cluster-based Network Services. In *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, Mar. 2001.

[29] P. Stok, M. Claessen, and D. Alstein. A Hierarchical Membership Protocol for Synchronous Distributed Systems. In *1st Europen Dependable Computing Conference*, LNCS 852, pages 597–616. Springer-Verlag, Oct. 1994.

[30] J. R. von Behren, E. A. Brewer, N. Borisov, M. Chen, M. Welsh, J. MacDonald, J. Lau, S. Gribble, and D. Culler. Ninja: A Framework for Network Services. In *Proc. of 2002 Annual USENIX Technical Conf.*, Monterey, CA, June 2002.

[31] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 1998.

[32] Yahoo! `http://www.yahoo.com`.