A Group Membership Service for Large-Scale Grids[*]

Fernando Castor Filho
Augusta Marques
Polytechnic School of Pernambuco
University of Pernambuco
{fernando.castor, armf}@dsc.upe.br

Raphael Y. de Camargo
Fabio Kon
University of São Paulo
{kon,rcamargo}@ime.usp.br

ABSTRACT

In this paper, we propose a decentralized group membership service that can be incorporated into existing grid middleware to make it more reliable. This service includes a flexible failure detector that adapts dynamically to changing network conditions and can be configured with a number of failure recovery strategies. Moreover, it disseminates information about membership changes (new processes, failures, etc.) in a scalable and efficient manner. We conducted a preliminary evaluation of the proposed service by simulating a grid with up to 140 nodes distributed across three domains separated by a wide-area network. This evaluation showed that the proposed service performs well both in the absence and in the presence of process failures.

Categories and Subject Descriptors: C.2.4 [Computer Systems Organization] – Distributed Systems.

General Terms: Reliability, Design.

Keywords: grid computing, failure detection, group membership.

1. INTRODUCTION

In spite of all its benefits, both accomplished and potential, grid computing [9] is still an active research area and there are several open problems that need to be addressed for it to attain more widespread use. One such open problem is fault tolerance. The computations performed by a grid often last for several days. Furthermore, after grid resources are reserved, it might take several hours or days before they are available. Failure of a grid node can thus render several days of work useless and require that the grid resources be reserved again, wasting resources that could be leveraged to perform useful computation. Making a grid fault-tolerant can save time and allow for more efficient use of the grid.

To recover from failures of some of its nodes or communication channels, a grid must first detect the failures and guarantee that its participants are aware of them and can take appropriate measures. Moreover, the grid should be capable of dealing with dynamically joining processes, both new ones and previously failed ones. In the distributed systems literature, the middleware service responsible for these tasks is often called the Group Membership Service [1], or Failure Detection Service [23]. Group membership services usually comprise at least three parts: (i) a membership management component, responsible for keeping track of correct and faulty processes; (ii) a failure detector, responsible for monitoring a subset of the processes in the group and detecting process failures as quickly and accurately as possible; and (iii) an information dissemination component, responsible for notifying processes about membership changes of which they might not be aware. Group membership management is very important for large grids. It can speed up failure recovery and avoid inconsistencies in the results of long-term computations.

Due to the intrinsic characteristics of large scale grids, including the opportunistic ones [10], whose workstations can be shared with local users and only take part in the grid when idle, a group membership service should satisfy several requirements: (i) scalability, as a grid might comprise thousands of nodes; (ii) autonomy, because grids, besides being geographically distributed across potentially thousands of machines, might span multiple administrative domains and these factors make manual failure detection and recovery infeasible; (iii) capacity of handling dynamism; and (iv) efficiency in terms of consumption of network and machine resources; (v) distribution, in the sense that nothing short of a complete network outage should stop the operation of the group membership service.

In this paper we present the design and initial evaluation of a group membership service that aims to be practical and addresses the aforementioned requirements. This basic service was designed and implemented in a modular fashion and can be incorporated into existing grid middleware infrastructures and reused across several applications. Additionally, it is very lightweight (its implementation comprises less than 80 Kbytes of source code), runs on several hardware and software platforms, and is based on a well-established middleware technology. Furthermore, it is highly configurable and can be deployed in environments with different operational characteristics. The proposed group membership service leverages recent advances in epidemic (or gossip-style [23]) information dissemination protocols [5, 7] and ac-

[*] This research is supported by CNPq/Brazil, grants #481147/2007-1 and #550895/2007-8.

MGC'08 December 1-5, 2008, Leuven, Belgium.
Copyright 2008 ACM 978-1-60558-365-5/08/12 ...$5.00.

crual failure detectors [12, 19].

2. BASIC ASSUMPTIONS

For simplicity, we consider that a grid comprises a set of processes that communicate by exchanging messages. We do not make an explicit distinction between processes and nodes/machines in the grid and use these terms interchangeably. Also, we assume that any process can communicate directly with any other process via the underlying transport layer. We do not make any assumptions about the applications that will run on the grid nor about the programming models to which they adhere. We consider, however, that applications might require large amounts of bandwidth.

We assume a crash-recover model where processes fail silently but can recover from failures and rejoin the grid. We believe that, for opportunistic grids, this fault model is more appropriate than the crash model adopted by most works on gossip-style failure detection and group membership [13, 17, 23]. Given the large number of processes in a grid, we consider that failures are commonplace events and that some processes are failing and rejoining the grid at all times. It is even possible that a large percentage (50% or more) of the process in the grid fail simultaneously, due to events such as power shortages, network partitions, and Internet Worms [12]. Process failures can occur due to a number of reasons: hardware malfunctioning, application crashes, operating system crashes, a local user abruptly reclaiming the resources of a grid node, etc. For failure detection purposes, all these situations are identical.

Since we use TCP for communication, we consider that the underlying communication channels do not lose messages. Also, channels are partially synchronous [6], i.e., eventually, there will be a limit to the time it takes for a message to reach its destination. These assumptions attempt to mimic the way the Internet works and are in conformance with other works on failure detection [4, 13, 23].

3. BACKGROUND

This section provides a brief explanation about epidemic information dissemination and accrual failure detectors.

Epidemic Information Dissemination. Epidemic- or gossip-style information dissemination protocols were proposed in the late 1980's to improve the reliability and scalability of updates to replicated databases and have received considerable attention in the distributed systems community in the last 10 years. They mimic the way in which infectious diseases (or gossip) spread. In a gossip protocol, when a process P must disseminate some information to other processes, it chooses some of them randomly and sends the information. Each process Q that receives the information merges it with any up-to-date information that it has and repeats this procedure, randomly selecting some processes and sending the merged information. Van Renesse et al. [23] have shown, in a theoretical study, that the information will get to all the processes in the group with a very high probability (almost 1) if each process gossips to $c * \log N$ processes, where c is a constant and N is the number of processes in the group. Several works have demonstrated both in theory and in practice that gossip protocols are both robust and scalable [4, 7, 13, 17, 23]. Examples of applications of gossip protocols include failure detection [23, 13], group membership [4, 17], and replicated database updates [5].

A gossip protocol can be reactive or periodic [17], depending on whether speed of information dissemination or low bandwidth consumption is more important. In the former case, upon receipt of a new gossip message, a process P chooses some targets and gossips to them. In the latter, each process gossips some relevant information to a number of randomly chosen processes at regular time intervals. Also, gossiped information might be sent on its own messages or piggybacked on other messages [4], again depending on whether speed or bandwidth consumption is more important. Additionally, gossip protocols can use TCP or UDP as transport protocol [13]. On the one hand, TCP requires more bandwidth and includes mechanisms, such as flow control, that are undesirable for some applications. Nevertheless, it provides delivery guarantees and is firewall-friendly. On the other hand, UDP is more lightweight but provides no guarantees and is often blocked by firewalls. Finally, the reliability and speed of dissemination of a protocol depend on its *fanout* value [7], the number of processes to which each process gossips information. Gossip protocols, due to their inherent redundancy, can often achieve high levels of reliability and speed of dissemination using small fanout values.

Accrual Failure Detectors. Accrual failure detectors [12] are a solution to the inherent lack of flexibility of traditional heartbeat-based failure detectors. In a traditional failure detector, if a process P monitors a process Q, the latter periodically sends heartbeat messages to the former. If a certain amount of time T_{to} elapses without P receiving a heartbeat from Q, Q is considered to have failed. Some schemes adjust the value of T_{to} dynamically to adapt to changing network conditions [3].

Accrual failure detectors use heartbeat inter-arrival times as samples to define a probability distribution and, considering the time since the last heartbeat was received, compute the probability that the monitored process has failed. As a result, they naturally adapt to changing network conditions while completely hiding the notion of time from their clients. Since accrual failure detectors output a suspicion value on a continuous scale (instead of simply declaring a process as failed or not), it becomes possible to define several distinct failure recovery strategies based on the level of suspicion that a process has failed. This feature results in greater flexibility for system administrators.

4. GROUP MEMBERSHIP SERVICE

The proposed group membership service has three main components: (i) a Failure Detector component , that detects failures of group members; (ii) an Information Dissemination component, that disseminates information about joining or failing members; and (iii) a Membership Management component, that maintains membership information. Figure 1 depicts the high-level architecture of the group membership service module running within each node in the group. In the rest of this section we explain each of these components in more detail.

4.1 Membership Management

The Membership Management component keeps track of the processes that each process knows about, the ones it believes to have failed, the ones that monitor it, and the ones that it monitors. Moreover, this component coordinates the

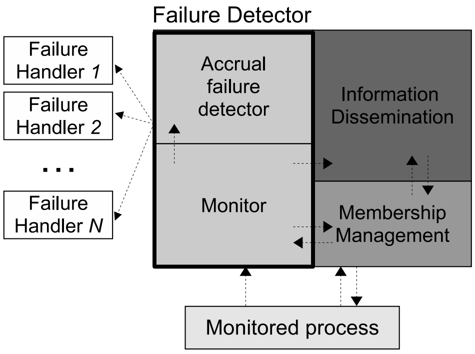


Figure 1: Main components of the group membership service. The dashed arrows indicate the direction of the interactions between components.

interaction between the Failure Detector and Information Dissemination components. When a new process Q wants to join the group, it must issue a `monitor_req` request to a process P that is already a member of the group. P, upon receipt of the monitoring request, will always answer positively with a `monitor_ack` response. P will also gossip to Q the IDs of some processes it knows. Processes in the group membership service are uniquely identified by an <`IP`>:<`PORT`> pair. The number of processes about which P gossips to Q is dictated by parameter L. In most of our experiments, we have set L to 12. Furthermore, with a probability inversely proportional to the number of processes that P knows, it will also request Q to monitor it.

Each process in a group can be monitored by at most K other processes, i.e., larger values of K mean that the monitoring network is more resilient to failures. Setting K to a small integer value, such as 5 or 6, is enough for most practical purposes [13, 17]. A process R which is monitored by a process S has an open TCP connection with it through which it sends heartbeat messages (Section 4.2) and other kinds of information. If R perceives that it is being monitored by more than K processes, it issues a `monitor_cancel` message to a randomly chosen process and cancels the monitoring relation, so as to keep the number of monitoring connections as close as possible to K.

When a process either detects (Section 4.2) or gets to know about (Section 4.3) the failure of a process f, it removes f from its set of known processes. However, in practical scenarios, specially in opportunistic grids, a failed process will eventually reappear and attempt to rejoin the grid. Hence, upon declaring a process as failed, each process initiates a timer whose length is user-defined. The timer must be long enough for the failure information to spread to all the group processes. If f attempts to rejoin before the timer expires, its monitoring request is refused. After timer expiration, each process removes f from its set of failed processes and informs neighboring processes about this. From this moment on, the previously-failed process can attempt to rejoin the grid.

4.2 Failure Detector

The Failure Detector component comprises the Monitor and the Accrual Failure Detector (AFD) components (Figure 1). The Monitor component collects information about the health of the monitored process. This information consists of heartbeat messages that monitored processes periodically send to the monitoring processes. Parameter T_{hb} determines the time between two consecutive heartbeats sent by a monitored process. As mentioned in Section 4.1, monitoring relationships are established in a random fashion. This is very important to ensure the robustness and reliable message delivery properties of gossip protocols [23, 4]. In the proposed group membership service, monitored processes choose monitoring processes randomly, but always send heartbeat messages to the same processes, as long as the monitoring relationship lasts. This approach ensures that the accrual failure detectors collect information about the monitored processes in a timely fashion. Previous work has shown that maintaining persistent monitoring relationships does not compromise the properties of gossip protocols, as long as the monitoring relationships are randomly established [13, 17].

Upon receipt of a heartbeat, the Monitor component obtains the elapsed time since the previous heartbeat was received and provides this information to the AFD component. The AFD keeps a record of the last $NUM\_SAMPLES$ samples it received, where $NUM\_SAMPLES$ is a user-defined parameter. Similarly to other works on accrual failure detectors [12, 19], we normally set $NUM\_SAMPLES$ to 1000. At each clock tick (typically every 0.1 second), the Monitor asks the AFD to check for failed processes. The AFD calculates a suspicion value based on the formula below [19]:

$$P_{fail,S}(t_\Delta) = \frac{|S^{t_\Delta * \alpha}|}{|S|} \qquad (1)$$

where S is the set of all received samples, $|S|$ is the number of elements in S ($|S| \leq NUM\_SAMPLES$), t_Δ is the time elapsed since the last heartbeat was received, α is a scaling factor ($0 < \alpha \leq 1$) that trades off failure detection time and false positive rate [2], and $|S^{t_\Delta * \alpha}|$ is the number of samples that are smaller than or equal to $t_\Delta * \alpha$. The rationale is that the larger the number of samples smaller than $t_\Delta * \alpha$, the lower the probability that a new sample will arrive. We use an optimized implementation based on an extended AVL tree to calculate the failure probability using at most $O(\log |S|)$ comparisons. Hence, even if $NUM\_SAMPLES$ is very large (e.g., 1,000,000) and suspicion values are requested every 0.1 second, their calculation is kept computationally inexpensive.

The AFD component of each process begins its execution without any information about the monitored processes. In this initial state, we want it to behave as a traditional timeout-based failure detector until is has acquired enough information, otherwise it might produce too many false positives. At the same time, we do not want to modify its design or implementation to achieve this behavior. We satisfy these two requirements by providing a set of dummy samples to the AFD component at startup. Each dummy sample has a value dictated by parameter T_{to}, provided by the system administrator, and chosen so as to be enough to avoid too many false positives while achieving reasonable failure detection time. To the best of our knowledge, we are the first to propose this modification to existing accrual failure detection schemes [12, 19]

The AFD component can be configured to execute several different failure handlers. Handlers are associated with thresholds that define probabilities that a process has failed. When the suspicion that a process P has failed reaches one of the established thresholds, the corresponding failure handler is triggered by the AFD. Notice that the AFD goes on monitoring P. Hence, if the suspicion that P has failed reaches a higher threshold, the AFD will trigger the associated handler and this continues until one of the handlers removes P from the monitoring list of the AFD component. With this approach, preventive failure recovery actions can be defined for lower threshold values, whereas more energetic procedures can be enacted for higher ones. The Group Membership Service includes a default failure handler that removes the failed process from the list of monitored processes of the AFD and uses the dissemination component (Section 4.3) to notify other processes about the failure.

Previous experiments [12, 19] have shown that accrual failure detectors perform similarly to timeout-based adaptive failure detectors [3] when heartbeat inter-arrival times follow a normal distribution. In our experiments, though, we have noticed that they do not perform so well when network latencies are regular during long periods of time punctuated by transient and short high-latency outbursts. This tends to produce false positives unnecessarily. To mitigate this problem, we use a very simple suspicion mechanism where the monitoring process attempts to send a heartbeat to the monitored process. If the heartbeat is successfully sent, the former stops suspecting the latter. Otherwise, the monitored process is considered failed. Moreover, if the heartbeat successfully reaches the monitored process but the accrual failure detector starts suspecting the monitored process again before receiving a heartbeat, the process is considered failed. The rationale for this approach is that, sometimes, an application might fail while its infrastructure (operating system, middleware platform, etc.) goes on working. In this scenario, it is not possible to detect the problem by simply sending a heartbeat in the opposite direction.

4.3 Information Dissemination

The Information Dissemination component notifies other processes about membership changes and, more generally, the current members of the group. The main choices to make, when designing a gossip-based information dissemination component are (i) whether it will use a periodic [17] or reactive approach [13], (ii) what information will be gossiped, and (iii) what will be the speed of dissemination. In this work, we have let the main needs of computational grids guide our decisions. First of all, in a grid, information about failed processes should be disseminated as fast as possible, in order not to hinder progress, avoid general application failures, or, in cases where a general failure is inevitable, allow the grid job to be restarted (or rescheduled) as soon as possible. Second, information about new members is not so urgent, albeit useful. The rationale, in this case, is that not knowing the complete group membership or newly joining processes does not hinder progress, whereas not knowing about a failed process might.

To disseminate failure information, the group membership service uses a reactive and explicit approach. This means that once a process learns about a new failure it automatically sends this information to J processes that it monitors or that monitor it. The administrator-defined parameter J dictates the speed of dissemination. As mentioned in Section 3, for some information gossiped by a single process to reach all the group members with high probability, J should be $\simeq c * \log N$. On the other hand, no explicit action is taken to disseminate information about new processes. Instead, processes get to know about new processes by simply receiving heartbeat messages. Each heartbeat that a process p sends to a process q includes some randomly chosen ids of K (Section 4.1) processes that p knows about. Piggybacking process ids in heartbeat messages has the following consequences: (i) network load is reduced, since no new messages are generated and the information spreads throughout the group at a slower pace; (ii) the size of a heartbeat is bounded to K, a constant that does not change at runtime and whose value is usually $\simeq \log N$; and (iii) dissemination time becomes less predictable because, due to the random selection of process ids, it might take a long time before the id of a new process is gossiped.

4.4 Implementation

Our group membership service is implemented in Lua [15], an extensible and lightweight programming language. Lua makes it easy to use the proposed service from programs written in other programming languages, such as Java, C, and C++. Moreover, it executes in several platforms. Currently, we have successfully run the failure detector in Windows XP, Mac OS X, and several flavors of Linux. The entire implementation of the group membership service comprises approximately 80Kb of Lua source code, including comments. Moreover, for interprocess communication, we use a CORBA ORB named OiL [18] which is also written in Lua. OiL is multi-platform, very lightweight, and performs well when compared to existing production ORBs [18].

The implementation of the group membership service is modular. The entire service is encapsulated within a Lua module, which means that it is easily accessible to applications. To use the proposed service from a Lua program, only two lines are necessary:

```
gfd = require ("gfd");       -- imports the service.
gfd.start(id, known_hosts); -- starts the service.
```

In the code above, `id` is the identifier of the newly started process and `known_hosts` is a list consisting of identifiers of grid processes that it knows *a priori*.

5. EVALUATION

This section presents a preliminary evaluation of the proposed group membership service. The evaluation consisted of executing several instances of the service (from 20 to 140) within three 2.53 GHz machines with 1GB RAM each and communicating through a 100Mbps Fast Ethernet local area network. Moreover, we have simulated a wide-area network using the WANEem [22] emulator. WANem is a parametric emulator that supports the configuration of several parameters of wide-area networks, such as latency, jitter, % of duplicates, message loss, etc. We have set two of these parameters, latency and jitter, to 500ms and 250ms. During the experiments, the network experienced light traffic and an average packet loss rate of 3% (as measured by the ping Unix command). We ran experiments involving 20, 40, 60, 80, 100, 120, and 140 instances of the group membership service (processes). In the first five cases, we employed two machines, each running half the overall number of processes. In the latter two experiments, we employed three machines, two running 50 processes each and the third running 20 and 40 processes, respectively. All the traffic between different machines in all the experiments passed through WANem, thus simulating a grid comprising three medium-sized clusters separated by the Internet. For each process, we have set parameters T_{hb} to 2s, K to 4, and J to 6 (Section 4.2).

The main goal of the experiments was to assess the scalability of the group membership service in terms of the number of messages sent per process. We have measured the average number of messages sent per process in two distinct situations: (i) when no failures occur; and (ii) when a percentage of the grid processes fail. In the first case, the number of processes ranged from 20 to 140, as described above. In the second one we fixed it at 140. In the latter case, we are specially interested in measuring the overhead that a reactive approach to disseminate failure information imposes. Also, we want to observe the resilience of the group membership service to failures of a large number of processes. Each experiment was conducted by initializing the processes at regular intervals (with a 2 second interval between two consecutive processes). Most instances of the group membership service received information about only one preexisting process. Moreover, up to 20% of the members of each cluster also received information about a single node located at a different cluster. We have recorded the number of sent messages per process after 7800 message exchange rounds.

Figure 2(a) presents the results of the experiments where no failures were injected in the simulation. The graph shows that the average number of messages sent per process varied only slightly with the growth of the number of processes. The highest average number of sent messages per process (in the experiment involving 100 processes) was only 2,97% higher than the lowest average number of sent messages per process (20 processes). Throughout the experiments, the number of sent messages did not exhibit a definite tendency to grow with the number of processes. This evidence suggests that the proposed group membership service might in fact be scalable and supports previous analytical results [4].

Figure 2(b) shows the effect that process failures have on the average number of messages sent per process. We have injected the failures all at once. This adverse approach is useful for evaluating the group membership service because it does not give it time to re-organize between failures, thus assessing its resilience. For a large number of failures (10% of all the processes in the grid), the average number of messages per process grew 4,02% when compared to the scenario where no failures occur. When an unreasonably large number of failures occurs (40% of the process), the number of messages grew 13,37%. Thus, we can say that the proposed service also scales well in the presence of process failures, specially if we consider that all the failures were injected simultaneously and during a short experiment. Moreover, in spite of the large number of failures, no process became isolated and no process was left unmonitored. More specifically, 77,4% of them were monitored by K other processes and 94,05% were still monitored by at least $K-1$ processes.

Finally, Figure 2(c) exhibits the average number of messages sent by the non-failed processes running within one of the clusters in the scenario where 40% of the processes fail. The failures were all injected between the 6000th and 6100th message exchange rounds. Reactive dissemination of failure information clearly injects a large number of extra messages in the network in a short amount of time. In this case, we have traded this off with faster information dissemination.

6. RELATED WORK

There are several proposals of approaches to make grids fault-tolerant. Amongst them, a large part describes mechanisms for monitoring and failure detection of grid nodes. Globus HBM [21] targets the Globus toolkit [8], leverages unreliable failure detectors [2], and allows users to establish a compromise between failure detection time and rate of false positives. Globus HBM disseminates information about failures by flooding and does not use adaptive failure detection. Therefore, its scalability and adequacy to opportunistic grids are limited. In a more recent work, Hwang and Kesselman [14] have devised a generic and flexible failure detection infrastructure for grids. The main advantage of this service, when compared to Globus HBM, is that it supports the definition of several distinct failure recovery policies. Nevertheless, it suffers from the same limitations as Globus HBM, in terms of scalability and adaptiveness.

Legion [11] employs a hierarchy of "phoenix" servers to monitor grid nodes and restart them when necessary. Monitoring adheres to a push model where monitored nodes periodically send heartbeat messages to the monitoring infrastructure. This work is based on the often unrealistic premise that monitoring nodes do not fail. In this scenario, information dissemination is a minor problem. Also, this approach does not adapt well to changing network conditions.

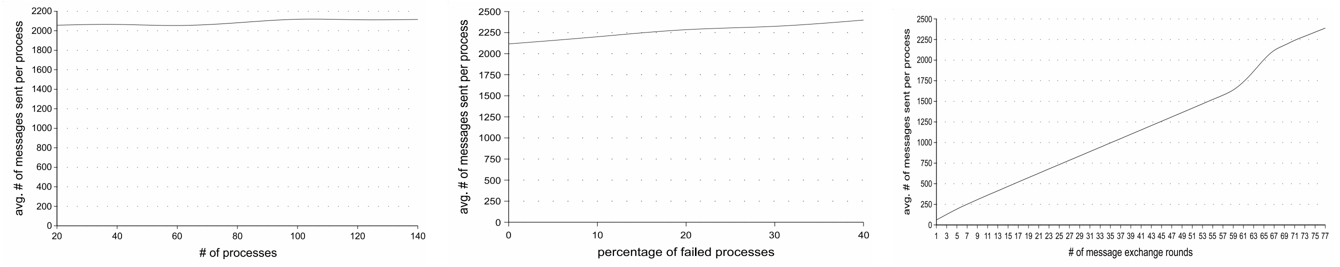
Shi and colleagues have introduced the ALTER architecture [20] for failure detection in grids. Its main highlight is the use of a timeout-based adaptive failure detector. Its main limitations are lack of flexibility, since it uses timeouts and its lack of integration with a group membership service. Group membership is managed by a separate index service that functions as a single point of failure for the system. Moreover, it is not clear how scalable ALTER is.

The work of Jain and Shyamasundar [16] is directly related to our own. The authors describe a failure detection and membership management service for grids centered around the concept of heartbeat groups. Heartbeat groups prescribe a hierarchical organization for failure detectors that improves scalability at the cost of losing some reliability, since each group is coordinated by a centralized leader (assisted by a single backup process). This approach does not leverage epidemic dissemination nor uses adaptive failure detection to handle changing network conditions. Moreover, it does not support the flexible association of failure recovery strategies to failure thresholds.

7. CONCLUDING REMARKS

We have presented a proposal for a group membership service for large-scale grids where nodes fail by crashing and subsequently recovering. This proposal combines recent advances in gossip-style information dissemination and accrual failure detectors to produce a service that is, at the same time, scalable and flexible. Preliminary results suggest that the proposed group membership service is in fact scalable, both when no failures occur and when a large number of processes fail simultaneously.

Even though accrual failure detectors are often advertised as flexible approach to failure detection, to the best of our knowledge, we are the first to design and implement an accrual failure detection service where users can associate several different failure handlers with different failure thresholds. Moreover, we have introduced a small improvement to help the failure detector to behave properly when it still has not collected enough information. Finally, existing



(a) (b) (c)

Figure 2: Results of the experimental evaluation.

works [12, 19] do not consider how AFDs obtain information about monitored processes, even though an AFD needs a considerable number of heartbeats to work properly. We have shown that gossip-based dissemination can be used to support scalable AFDs, as long as processes always gossip within the same set of randomly chosen processes.

In the future, we intend to further evaluate the proposed service. Besides conducting simulations with a larger number of nodes (300+), we would like to assess its performance when running within a real grid computing setting. Furthermore, we intend to compare it with other approaches for group membership, in terms of performance, reliability, and ease of configuration.

8. REFERENCES

[1] K. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, 1993.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[3] W. Chen, M. K. S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(5):561–580, May 2002.

[4] A. Das, I. Gupta, and A. Motivala. Swim: Scalable weakly-consistent infection-style process group membership protocol. In *Proc. of the 32nd International Conf. on Dependable Systems and Networks*, pages 303–312, 2002.

[5] A. Demers et al. Epidemic algorithms for replicated database maintenance. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 1–12, 1987.

[6] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[7] P. T. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, 2004.

[8] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputing Applications*, 2(11):115–128, 1997.

[9] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, 1999.

[10] A. Goldchleger et al. Integrade: Object-oriented middleware leveraging idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16(8):449–459, March 2004.

[11] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Legion: an operating systems for wide-area computing. Technical Report CS-99-12, University of Virginia, March 1999.

[12] N. Hayashibara, X. Defago, R. Yared, and T. Katayama. The phi accrual failure detector. In *Proc. of the 23rd International Symposium on Reliable Distributed Systems*, pages 66–78, 2004.

[13] Y. Horita, K. Taura, and T. Chikayama. A scalable and efficient self-organizing failure detector for grid applications. In *Proceedings of the 6th ACM/IEEE International Workshop on Grid Computing*, 2005.

[14] S. Hwang and C. Kesselman. A flexible framework for fault tolerance in the grid. *Journal of Grid Computing*, 1(3):251–272, September 2003.

[15] R. Ierusalimschy, L. H. de Figueiredo, and W. C. Filho. Lua - an extensible extension language. *Software: Practice Experience*, 26(6):635–652, 1996.

[16] A. Jain and R. Shyamasundar. Failure detection and membership management in grid environments. In *5th International Workshop on Grid Computing*, 2004.

[17] J. Leitao, J. Pereira, and L. Rodrigues. Hyparview: A membership protocol for reliable gossip-based broadcast. In *Proc. of the 37th International Conf. on Dependable Systems and Networks*, June 2007.

[18] R. Maia, R. Cerqueira, and R. Cosme. OiL: An object request broker in the Lua language. In *Proc. 24th Brazilian Symposium on Computer Networks*, 2006.

[19] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *Proc. of the 22nd ACM Symposium on Applied Computing*, 2007.

[20] X. Shi et al. Alter: Adaptive failure detection services for grids. In *Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 355–358, June 2005.

[21] P. Stelling, I. T. Foster, C. Kesselman, C. A. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *Proc. of the 7th International Symposium on High Performance Distributed Computing*, pages 268–279, 1998.

[22] Tata Consulting Services. Wanem - the wide area network simulator. Last visit: August 2008 – http://wanem.sourceforge.net/.

[23] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of Middleware'1998*, September 1998.