A Notification System for the φ-Failure Detector

Naohiro Hayashibara† Xavier Défago‡,* Makoto Takizawa†

†School of Science and Engineering,
Tokyo Denki University (TDU)

‡School of Information Science,
Japan Advanced Institute of Science and Technology (JAIST)

*"Information and Systems," PRESTO,
Japan Science and Technology Corporation (JST)

E-mail: {haya,taki}@takilab.k.dendai.ac.jp, defago@jaist.ac.jp

Abstract

It is widely recognized that distributed systems would greatly benefit from the availability of a generic failure detection service. There are however several issues that must be addressed before such a service can actually be implemented.
In this paper, we highlight the issue related to propagating information on suspected processes. Traditionally, failure detection systems provide information on suspects to every processes. However, it is not the efficient way if the system has lots of nodes and processes. We now propose a notification system that propagates information on suspicions with content-based filtering. It can provide information to proper receiver and separate the task to send such information from failure detector modules.
Therefore, the notification system can reduce the amount of information that should be sent. each failure detector module would be light-weight because it just provide own information on suspected nodes and processes to the notification system.

1 Introduction

The ability for a distributed system to detect the failure of its processes is widely recognized as an essential issue for fault-tolerant systems. In fact, virtually any practical fault-tolerant distributed application relies on a form of failure detection mechanism or another to react appropriately in the face of failures. In such applications, failure detection can be invoked either directly, or indirectly through the use of a group membership service or other group communication primitives (e.g., consensus, total order broadcast).

Our objective is to implement and provide a generic failure detection service for large-scale distributed systems. The idea of providing failure detection as an independent service is not itself particularly new (e.g., [3, 6, 14, 15]). However, several important points remain to be addressed before a truly generic service can be proposed.

Specially, mechanisms for finding suspicious processes have been developed very well so far. While, only few have been developed for propagating information on suspected processes (e.g., [14, 15, 6]).

In this paper, we discuss on the way for propagating information of failures in the φ failure detector. In this context, we have to consider the way to propagate suspicion levels, which are continuous value and changed over time. Also, the information should be sent to proper destinations. It means that useless information shouldn't be sent by the propagation protocol and the system. Information of a certain process's failure is needed by some specific processes. It means that some others do not need this information. Thus, notification of failure information is looks like some form of publish/subscribe system.

The remainder of the paper is constructed as follows. In the section 2, we describe the target system of the paper and some related works. In the section 2.2, we introduce several failure detectors and existing propagation techniques. Then we discuss on the design of the notification system and the interaction between the system and failure detector modules in the section 3.

2 System Model and Related Works

2.1 System Model

We represent a distributed system as a set of processes $\{p_1, p_2, \ldots, p_n\}$ which communicate only by sending and receiving messages. We assume that every pair of processes is connected by two unidirectional quasi-reliable communication channels [1]. A quasi-reliable channel is defined as a communication channel which guarantees (1) no message loss, (2) no message corruption, and (3) no creation of spurious messages. We consider that processes may only fail by crashing, and that crashed processes never recover.

We assume the system to be asynchronous in the sense that there exist bounds neither on communication delays nor on process speed. For each communication channel, we assume message delays to be determined by some random variable whose parameters are unknown, independent of other communication channels, and whose distribution is positively unbounded. We assume that the parameters of the random variable can change over time, but that they eventually become stable.

2.2 Related Work on Failure Detection

In this section, we briefly introduce some of the most important concepts related to failure detection in distributed systems. Firstly, we introduce the theoretical foundation of failure detection (§2.2.1). Secondly, we present some of extended solutions for various environments and/or large number of processes that should be monitored (§2.2.2). Thirdly, we discuss a second aspect of failure detection services: the notification of failures (§3).

2.2.1 Unreliable Failure Detectors

Chandra and Toueg [3] define failure detectors as a distributed oracle with well-defined properties. A failure detector is a distributed entity which consists of a set of failure detector modules, one attached to each process. A failure detector module FD_p, attached to a process p, maintains a set of suspected processes. Process p can query its failure detector module at any time. Whenever some process q appears in the set maintained by FD_p, we say that p *suspects* q (that is, p suspects that q has crashed). The failure detector is however unreliable in the sense that its modules are allowed to make mistakes (1) by erroneously suspecting some correct process (wrong suspicion), or (2) by failing to suspect a process that has actually crashed. A module can also change its mind, for instance, by stopping to suspect at time $t+1$ some process that it suspected at time t.

2.2.2 Adaptive Failure Detectors

Adaptive failure detection mechanisms are designed to adapt dynamically to their environment and, in particular, to adapt their behavior to changing network conditions. Failure detectors can also be made to adapt to changing application behavior. They are basically extension of the notion on failure detectors proposed by Chandra and Toueg. Roughly speaking, they can adjust a timeout according to the network condition. Deciding suspicions is done by the same way as the traditional one introduced above.

Adapting to network conditions There exist several propositions of adaptive failure detection mechanisms (e.g., [2, 4, 7, 13]). The proposed solutions are based on a heartbeat strategy, although nothing seems to preclude the use of other strategies such as interrogation. The principal difference with the heartbeat strategy is that the timeout is modified dynamically according to network conditions.

Chen et al. [4] propose a different approach based on a probabilistic analysis of network traffic. The protocol uses arrival times sampled in the recent past to compute an estimation of the arrival time of the next heartbeat. The timeout is set according to this estimation and a safety margin, and recomputed for each interval. The safety margin is determined by application QoS requirements (e.g., upper bound on detection time) and network characteristics (e.g., network load).

Bertier et al. [2] propose a different estimation function, which combines Chen's estimation with another estimation of arrival times developed by Jacobson [12] for a different context.

Adapting to application requirements Some of the adaptive failure detectors mentioned above [4, 2] can be tailored to match diverse applications requirements. This is done by using QoS requirements to compute the parameters of the failure detector. Then, the failure detectors adapt to changing network conditions in such a way that the QoS requirements are met with high probability.

The main drawback of the failure detectors mentioned above is that they are designed with one single application in mind. This means that, even if parameters can be adjusted to match QoS requirements, they can only meet those of one single application at a time. Arguably, QoS requirements could be set as a least common factor of all concurrent applications. However, this is unfortunately not that simple in practice, as doing only results in tradeoffs that are impossible to address.

2.3 Related Work on Propagation Techniques with Failure Detectors

2.3.1 Tree-based Protocol for Globus

Stelling et. al. [14] propose a failure detection service for the Globus Grid toolkit, a middleware platform to support Grid applications [8]. The architecture of the proposed failure detector is based on two layers. The lower layer consists of *local monitors*, while the upper layer consists of *data collectors*. A local monitor is responsible for monitoring all processes that run on the same host. The local monitor periodically sends heartbeat messages to data collectors, including information about the monitored processes. Data collectors gather heartbeats from local monitors, identify failed components, and notify the applications about the suspicions.

2.3.2 Gossip-based Protocols

Gossip-style failure detectors [15, 9], also sometimes called epidemic-style failure detectors, are based on the observation that rumors (or diseases) can propagate very efficiently within a system, even a very large one. More concretely, failure detector modules pick random partners with whom they exchange information about suspected processes. Doing so ensures that suspicions are eventually propagated over the whole system.

One of the very strong advantage of gossip-style protocols is that they are completely oblivious of the underlying topology, and hence are completely oblivious to topology changes. In other words, topology changes do not need to affect the performance of this class of failure detectors.

3 Notification System for the φ Failure Detector

In practice, failure detection service should play two fundamental roles: suspecting when monitored processes fail, and conveying this information to the monitoring processes. In local networks, these two roles are combined. This is not the case in large-scale distributed systems, where the two aspects should be distinguished. Doing so allows to ensure that the detection of failures remains a local mechanisms, whereas the distribution of failure suspicions is done by some notification mechanism.

We now focus on the design of a notification system for the φ failure detector. It means that the system helps this type of failure detector to propagate suspicion levels to proper destinations.

3.1 The φ Failure Detector

Towards a generic failure detection service, we need to build a scalable and precise failure detection mechanism because failure detector modules are parts of the service. However, lots of problems lay on this goal [10] and no solution has succeeded completely so far. Now we focus on the ability of adapting to network condition and requirements of processes for failure detection.

To address the problems mentioned above, we have recently developed a novel approach to failure detectors, called the φ-failure detector [11]. φ-failure detectors are based on the notion of *Accrual failure detectors* [5], which use no timeout and reconcile all three types of adaptation. The key idea is that a φ-failure detector provides information on the degree of confidence, called *suspicion level*, that a given process has actually crashed. More specifically, the failure detector associates a value φ_p to every known process p. The value φ_p increases dynamically according to a normalized scale and represents the degree of confidence, at the time of query, that process p has crashed.

The interactions between the applications and the failure detector are hence different than in the traditional case. Indeed, distributed applications use the value φ_p associated with a process p to decide on a course of action. For instance, applications can set some finite threshold for φ_p and decide to suspect p if φ_p crosses that threshold. Different applications can then set different thresholds for the same process. For instance, some applications would set a low threshold to obtain prompt yet inaccurate failure detection (i.e., with many wrong suspicions), while applications with stronger requirements would set a higher threshold and obtain more accurate suspicions. Consequently, this approach can effectively adapt to application requirements because the threshold can be set on an per-application basis (and also on a per-communication channel basis within each application). Beside, the scale ensures that the value set as a threshold is meaningful for the application (it represents the degree of confidence). In practice, we compute the value φ_p based on the history of arrival intervals between heartbeat messages (see [11] for details).

Let us illustrate with a simple example what we describe as the adaptation to application requirements. Consider for instance two applications A_{in} and A_{db}, where A_{in} is an interactive application and A_{db} is a heavyweight database application. Consider also than both applications run simultaneously and rely on the same system-wide failure detection service. With A_{in}, the actual crash of a process must be detected quickly to prevent the system from blocking. In contrast, A_{db} launches a multi-terabytes file transfer whenever a process is suspected, and hence requires accurate suspicions. While A_{in} favors the reactivity of the failure detector, A_{db} requires high accuracy.

On the view of propagating information given by the φ-failure detector, we can not use the exist mechanisms introduced in the Sect. 2.3, because no one consider to disseminate suspicion levels, which are continuous values changed over time. This problem is a serious for the φ-failure detector. It can realize the adaptability for network condition and application requirements by

many processes. Each failure detector module outputs suspicion level on a certain process. Suspicion level means how much chance to get a correct suspicion if the failure detector module suspects the process at certain time. It is also represented as a positive real value. It means that this information should be delivered as soon as possible by application processes that are interested in it.

Thus, we now discuss on the content-based propagation of information given by failure detector modules. Propagating information based on requirements were investigated in the context of multicasting and group communication. While it is also studied as *publish/subscribe* interaction model.

In the viewpoint from application processes, normally they don't need information about every process. They are interested only in some specific groups of processes. In this case, the notification system has to provide information matched to their interests. This kind of interaction looks like publish/subscribe model.

3.2 Design of The Notification System for the φ Failure Detector

In this section, we describe the overview of the notification system for the φ failure detector. It maintains suspicion levels given by failure detector modules in a database and responds for inquiries by failure detector modules properly.

The notification system has three types of tables in a database to manage information on hosts and their processes as follows.

- Monitored Host table: It contains a host identifier (e.g., IP address), latest reported suspicion level, mean of suspicion level reported so far, smallest threshold registered so far and the number of monitors in each line of the table.
- Process tables: It could be made for each host and contains a list of processes running on a host with network connection. Each line has a process identifier (e.g., httpd, mountd), status flag (boolean value).
- Monitoring Host tables: It could be made for each monitoring host/failure detector module. Each of them contains a process identifier with a host identifier that it's running on and its threshold.

The system updates the database whenever a failure detector report information on hosts or processes. While, it starts to send notification messages when the smallest threshold of some host in the Monitored Host table.

3.3 Interaction with The Notification System

The propagation of information should be separated from failure detection in large-scale systems. The system plays a role that it propagates information on suspicions as events to proper receivers. Failure detector modules ask the notification system to propagate information on a certain process's failure. Therefore, the service should lay among failure detectors.

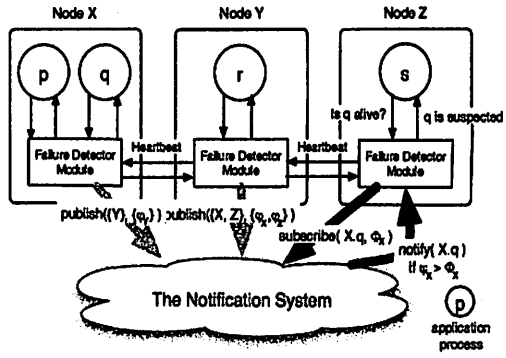

Figure 1. The design of the notification system for the failure detection service

The notification system also has the similar APIs, as ones in the publish/subscribe system (e.g., `subscribe()`, `notify()`), to propagate suspicion levels.

A failure detector module at a host H_i periodically provides suspicion levels on the set of hosts $\{H_1, ..., H_n\}$, which the module is currently monitoring, using the API `publish`$(\{H_1, H_2, ..., H_n\}, \{\varphi_{H_1}, \varphi_{H_2}, ..., \varphi_{H_n}, H_i\})$ given by the system. This event includes host identifiers and corresponding suspicion levels, which are computed just before the occurrence of the event. Host identifier is represented, for example, by IP address. In only the first step, a failure detector module could get a list of network connected processes for each host. Then, it sends the lists to the notification system to make process tables for each host.

We assume a node is monitored by several nodes (failure detector modules). It means that a failure detector module at a host sends heartbeat messages to several failure detector modules. Thus, several failure detector modules can provide different suspicion level on the node.

When a failure detector module at H_i detects that some process p_j has stopped while its execution, it sends a state change event to the notification system using the API `statechange`(H_i, p_j). Then, the notification system change the flag of the process and sends notification events to proper destination by using the registration host tables by using `notify`$(H_i.p_j)$. It would be also sent if a threshold of host H_i given by some failure detector module has be smaller than a reported suspicion level of H_i.

While, application processes can ask to the failure detector module in the local host about the sta-

tus of processes which they want to know. Then, the failure detector module can use the API subscribe to register a monitoring requirement for some processes or nodes to the notification system (e.g., If a failure detector module has requirements to monitor process p_1 at H_2 with $\Phi_{H_2.p_1}^{H_{sender}}$ and process p_3 at H_4 with $\Phi_{H_4.p_3}^{H_{sender}}$, it would issue `subscribe`($\{H_2.p_1, H_4.p_3\}$, $\{\Phi_{H_2.p_1}^{H_{sender}}, \Phi_{H_4.p_3}^{H_{sender}}\}$), where it includes target host identifiers for monitoring and corresponding desired threshold [1].) The system also has a facility of filtering information based on their contents. Thus, it can provide required suspicion levels to application processes.

Failure detector modules could receive a notification event whenever the registered threshold is exceeded. This event is also delivered by a failure detector module to each application.

When no application process wants to have any information on process p_j at the node H_i, the failure detector module could issue `unsubscribe`($H_i.p_j$). This API suspends to notify them such information/events.

Therefore, the notification system sends events to proper destination whenever the threshold is exceeded by the suspicion level or changing process's state.

Failure detector modules can also adjust the threshold of H_i, which has been registered, by using `check`(H_i). This API returns the mean of suspicion levels reported to the notification system, so far. Failure detector modules can get the mean value and change the threshold, to be grater than the previous one, with the subscribe API. Then, the notification system could override the new threshold in the database.

4 Conclusion

In this paper, we described the design of the notification system for the φ failure detector. Then, we discussed the way of propagating information in the φ failure detector with the proposed notification system. This approach can deliver desired information to appropriate failure detector modules immediately. Moreover, it can reduce the amount of messages sent by failure detector modules and the notification service. Because the notification system can provide information only to the failure detector modules that require it, and ask failure detectors to suspend sending unnecessary information.

In the future direction of our work is to customize the existing publish/subscribe system for building the notification system and define APIs more precisely.

References

[1] A. Basu, B. Charron-Bost, and S. Toueg. Solving problems in the presence of process crashes and lossy links. Technical Report TR96-1609, Cornell University, USA, Sept. 1996.

[2] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proc. of the 15th Int'l Conf. on Dependable Systems and Networks (DSN'02)*, pages 354–363, Washington, D.C., USA, June 2002.

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

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

[5] X. Défago, P. Urbán, N. Hayashibara, and T. Katayama. Definition and specification of accrual failure detectors. In *Proc. Int'l Conf. on Dependable Systems and Networks (DSN)*, Yokohama, Japan, June 2005. To appear.

[6] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Proc. 1st IEEE Intl. Symp. on Distributed Objects and Applications (DOA'99)*, pages 132–141, Edinburgh, Scotland, Sept. 1999.

[7] C. Fetzer, M. Raynal, and F. Tronel. An adaptive failure detection protocol. In *Proc. 8th IEEE Pacific Rim Symp. on Dependable Computing(PRDC-8)*, pages 146–153, Seoul, Korea, Dec. 2001.

[8] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *Intl. Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[9] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proc. 20th Annual ACM Symp. on Principles of Distributed Computing (PODC-20)*, pages 170–179, Newport, RI, USA, Aug. 2001. ACM Press.

[10] N. Hayashibara, A. Cherif, and T. Katayama. Failure detectors for large-scale distributed systems. In *Proc. 21st IEEE Symp. on Reliable Distributed Systems (SRDS-21), Intl. Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS'2002)*, pages 404–409, Osaka, Japan, Oct. 2002.

[11] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The φ accrual failure detector. In *Proc. 23nd IEEE Int'l Symp. on Reliable Distributed Systems (SRDS'04)*, pages 66–78, Florianópolis, Brazil, October 2004.

[12] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM'88*, Stanford, CA, USA, Aug. 1988.

[13] I. Sotoma and E. R. M. Madeira. Adaptation - algorithms to adaptive fault monitoring and their implementation on CORBA. In *Proc. of the Third Int'l Symp. on Distributed-Objects and Applications (DOA'01)*, pages 219–228, Rome, Italy, Sept. 2001.

[14] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A fault detection service for wide area distributed computations. In *Proc. 7th IEEE Symp. on High Performance Distributed Computing*, pages 268–278, July 1998.

[15] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In N. Davies, K. Raymond, and J. Seitz, editors, *Middleware'98*, pages 55–70, The Lake District, UK, Sept. 1998.

[1] Threshold represented as $\Phi_{H_i.p_j}^{H_k}$ means the failure detector module at host H_k requires to monitor a process p_j at host H_i with the threshold Φ