Two-ways Adaptive Failure Detection with the φ-Failure Detector

Naohiro Hayashibara* Xavier Défago*,† Takuya Katayama*

*School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Tatsunokuchi, Ishikawa 923-1292, Japan

†"Information and Systems," PRESTO,
Japan Science and Technology Corporation (JST)
E-mail: {nao-haya,defago,katayama}@jaist.ac.jp

Abstract

It is widely recognized that distributed systems would greatly benefit from the availability of a generic failure detection service. Such a service can however prove useful only if it can adapt simultaneously to changing network conditions and conflicting application requirements.

This paper presents a novel approach to adaptive failure detectors, called φ-failure detectors, which dynamically adapts to application requirements, as well as network conditions. The key idea is as follows. Traditionally, failure detectors maintain a set of suspected processes. The information is hence of boolean nature, that is, some process p is suspected if and only if it belongs to this set. In contrast, a φ-failure detector associates a value φ_p to every known process p. The value φ_p increases according to a normalized scale which represents the degree of confidence that process p has crashed. The scale is dynamically adapted from the current network conditions, and each application can trigger suspicions according to a threshold which corresponds to its own requirements. We describe a possible implementation for such a service, although some specific questions remain open where this work is still in progress.

1 Introduction

The ability for a distributed system to detect the failure of its processes is widely recognized as essential for fault-tolerance. In fact, almost every practical fault-tolerant distributed application must rely on a form of failure detection mechanism or another to react appropriately in the face of failures. In these 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 long-term goal is to define and implement a generic failure detection service for large-scale distributed systems. The idea of providing failure detection as an independent service is not particularly new (e.g., [3, 10, 19, 20]). Nevertheless, several important points remain to be addressed before a truly generic service can effectively be realized. In particular, a failure detection service must adapt to changing network conditions, as well as to application requirements. Several solutions proposed recently address the first issue specifically [2, 4, 11, 18]. However, to the best of our knowledge, none of the solutions proposed so far effectively address the second problem, namely, the adaptation to diverse application requirements. Rare propositions (e.g., [4]) have been made to adapt the parameters of a failure detector service to match the requirements, but are unfortunately designed to support a *single* class of requirements. Hayashibara et al. [13] have pointed this out recently in a short survey. So far, it seems that only Cosquer et al. [5] have identified the problem. Their proposition is interesting, but remains somewhat inflexible as they do not question the boolean nature of failure detection.

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.

The conflicting requirements mentioned above cannot possibly be reconciled by traditional timeout-based implementations of failure detectors. This is regardless of their

(c) 2003. Presented at *Proc. Workshop on Adaptive Distributed Systems (WADiS), pp.22–27, Sorrento, Italy, Oct. 2003*.

ability to adapt to changing network conditions, as shown by adaptive failure detectors described in the literature. Roughly speaking, these adaptive failure detectors are based on heartbeat messages and some timeout Δ_{to} determined dynamically. The value of Δ_{to} is computed with the recent history of message arrivals and an estimation function to predict the arrival of the next heartbeat. Using this approach, the failure detector can adapt to changing network conditions, but its accuracy is determined by the estimation function. Adapting to the requirements of several applications would require to manage as many timeout values, which is of course not acceptable. An alternate approach would allow applications to set their own timeouts, with the obvious drawback that failure detection cannot easily adapt to changing network conditions.

In this paper, we propose a different approach, called φ-failure detectors, which uses no timeout and reconciles both types of adaptation. The key idea is that a φ-failure detector provides information on the degree of confidence 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 according to a normalized scale and represents the degree of confidence that process p has crashed. The failure detector adapts to application requirements because each application can trigger suspicions according to its own threshold. Meanwhile, the failure detector can adapt to changing network conditions because the scale is defined accordingly.

The remainder of this paper is organized as follows. Section 2 presents the system model and some important definitions. Section 3 briefly describes traditional approaches to failure detection. In Section 4, we present the φ-failure detector, and describe a possible implementation in Section 5. Finally, Section 6 concludes the paper and discusses future directions.

2 System Model and Definitions

In this section, we describe our system model and remind important definitions. We briefly describe what failure detectors are. Then, we describe traditional implementations of failure detectors, followed by the more advanced adaptive failure detectors. We describe several recent variants of the latter.

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.

This system model is possibly a little stronger than the asynchronous model described by Fischer et al. [12] because we make some probabilistic assumptions on the behavior of the system. However, our model remains weaker than any of the partially synchronous models defined by Dwork et al. [9], because the fact that the distribution is positively unbounded implies that no bound (known or unknown) can exist on communication delays.

2.2 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, that p can query at any time. Whenever some process q appears in the set maintained by FD_p, we say that *p suspects q* (to have 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.

Several classes of failure detectors are defined according to two properties which restrict the mistakes that the failure detector can make. For instance, a failure detector of class $\Diamond\mathcal{P}$ must meet the following properties of completeness and accuracy.

Property 1 (Strong completeness) *Eventually every process that crashes is permanently suspected by every correct process.*

Property 2 (Eventual strong accuracy) *There is a time after which correct processes are not suspected by any correct process.*

The implementation of unreliable failure detectors is various system models has attracted significant work [17, 16, 15].

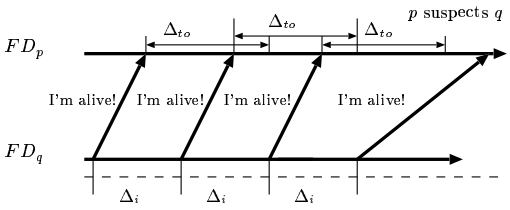

Figure 1. Heartbeat messages

3 Traditional Failure Detector Implementations

3.1 Heartbeat Strategy

The heartbeat strategy to implementing failure detectors is quite common. In this strategy, every failure detector module periodically sends a heartbeat message to the other modules, to inform them that it is still alive (see Fig. 1). The period is determined by the heartbeat interval Δ_i. A process p suspects a process q if FD_p, the module attached to process p, fails to receive any message from FD_q for a period of time determined by a timeout Δ_{to}.

There is the following tradeoff. If the timeout (Δ_{to}) is short, crashes are detected quickly, but the likeliness of wrong suspicions is high. Conversely, if the timeout is long, the chance of wrong suspicions is low, but this comes at the expense of the detection time. Beside, the fact that the timeout is fixed means that the failure detection mechanism is unable to adapt to changing conditions. This is because a long timeout in some system setting can turn out to be very short in a different environment. Beside, in practical systems, network conditions can greatly vary over time (e.g., depending on external load).

3.2 Adaptive Failure Detectors

Adaptive failure detectors address the problem of adapting to changing network conditions, that was mentioned previously. There exist many recent propositions of adaptive failure detectors [2, 4, 5, 11, 17, 18]. All of the solutions seen so far are based on a heartbeat strategy, although nothing seem to preclude the use of other strategies such as interrogation. The principal difference with the heartbeat strategy mentioned in Sect. 3.1 is that the timeout is modified dynamically according to network conditions.

The protocol proposed by Fetzer et al. [11] has a simple adaptation mechanism. It adjusts the timeout by using the maximum arrival interval of heartbeat messages. The protocol supposes a partially synchronous system model [9], wherein an unknown bound on message delays eventually exists. The authors show that their algorithm belongs to the class $\Diamond\mathcal{P}$ in this model.

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 QoS requirements (e.g. detection time) and network conditions (e.g. network load).

Bertier et al. [2] propose a different estimation function, which combines Chen's estimation with another estimation, due to Jacobson [14] and developed in a different context. Bertier's proposition provides a shorter detection time, but generates more wrong suspicions than when using Chen's estimation. The resulting failure detector is proven to belong to class $\Diamond\mathcal{P}$ in a partially synchronous system model.

Sotoma et al. [18] propose the implementation of an adaptive failure detector with CORBA. Their algorithm compute the timeout based on the average time for arrival intervals of heartbeat messages, and some ratio between arrival intervals.

The failure detector implementations proposed by Chen et al. [4] and by Cosquer et al. [5] can both be configured according to application requirements, but with some limitations. The former is designed to be dimensioned *statically* to a *single* application. The latter implementation can be tailored to several applications, but only to a limited extent and with a serious jump in the complexity of the failure detector.

Of all failure detector implementations mentioned in this section, the implementation proposed by Cosquer et al. [5] is the one which comes closest to addressing our problem. Yet, this does not come without practical limitations on the number of applications that can be served at any one time. Hence, we think that, in spite of their merit, they do not fully solve the problem of adapting to many application requirements. About other failure detection protocols, it can be said that they provide a "hardwired" degree of accuracy which must be shared by all applications.

3.3 Two Timeouts Failure Detectors

The two timeout approach advocated by Défago et al. [6, 7, 8] in earlier work can be seen as a first step toward adapting to application requirements, but the solution lacks generality. The two timeout approach was proposed and discussed in relation with group membership and consensus. In short, it was proposed to implement failure detection based on two different timeout values; an aggressive and a conservative timeout. The approach is well adapted for building consensus-based group communication systems. However, the protocol was not rendered adaptive to changing network conditions (although this would be feasible)

Table 1. The relation between φ and P_{acc}

φ	0	1	2	3	...	∞
P_{acc}	0	0.9	0.99	0.999	...	1

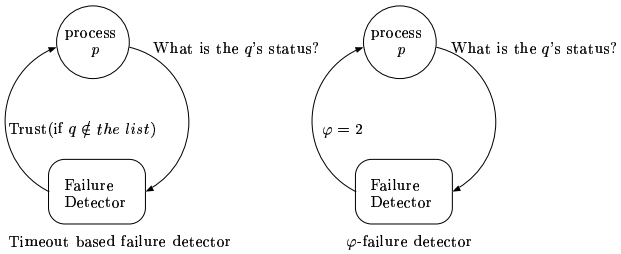

Figure 2. Timeout based failure detector vs. φ-failure detector

and, more importantly, lacks the necessary flexibility required by a generic service. Indeed, this could support only two classes of applications.

4 φ-Failure Detector

In this section, we propose the φ-failure detector. The principle is as follows. Instead of managing a list of suspected processes, each failure detector module associates a value $\varphi_p \in \mathbb{R}^+$ to every known process p. The value φ_p represents the degree of confidence that process p has crashed. This value is expressed according to a normalized scale, where $\varphi_p = 0$ means that there is no confidence at all about p, and $\varphi_p = \infty$ means absolute confidence that p has crashed. Thus modules maintain a list of pairs (p, φ_p) for every known process, and which can be queried anytime by all applications.

More precisely, the scale followed by φ_p is defined as follows. Let P_{acc} denote the probability that the statement "process p has crashed" will not be contradicted in the future (by the reception of a late heartbeat). Then, φ_p can be determined by Equation 1, which leads to the scale illustrated on Table 1.

$$\varphi_p = -log_{10}(1 - P_{acc}) \quad (1)$$

The failure detector must guarantee that φ_p increases monotonically between two periods where it is reset to 0.

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, the value P_{acc} can be computed based on the history of arrival intervals between heartbeat messages. In a possible implementation, a failure detector module analyzes the tendency of the system condition, network load and so on by the history of heartbeat messages.

5 Proposed Implementation

We propose to implement φ-failure detectors using a stochastic approach. These failure detectors have a history H of arrival intervals of heartbeat messages per a monitored process.

Each of the φ-failure detector has a window W as a set of histories of the processes $\{H_q, H_r, H_s, \ldots, H_z\}$. Let A_k be the arrival time for the k-th heartbeat message. The history H_q for a monitored process q is a sequence $\{A_1, A_2, A_3, \ldots, A_n\}$, where n is the number of received heartbeat messages, with entries sorted in increasing value.

The φ-failure detector modules analyze the tendency of heartbeat arrival time for some process q, based on H_q. H_q can be seen as a discrete distribution (see Fig. 3). Then, the failure detector module smoothes the distribution to get a continuous distribution, which allows to estimate P_{acc} at any given time. The module can use this distribution to compute P_{acc}, based on time t_{inq}, where t_{inq} is the time that a monitoring process p inquires the local failure detector module (see Fig. 4). P_{acc} is then converted into the φ scale defined in the previous section.

In fact, the curve in Figure 4 is positively unbounded, because the distribution includes a virtual message in the future. This means that there is always a possibility that the next heartbeat message will arrive at some future instant. Consequently, P_{acc} never reach 1 in an asynchronous system (see Table 1).

6 Discussion and Future Directions

Traditional adaptive failure detectors (see Sect. 3.2) can adapt to changing network conditions, based on a stochastic

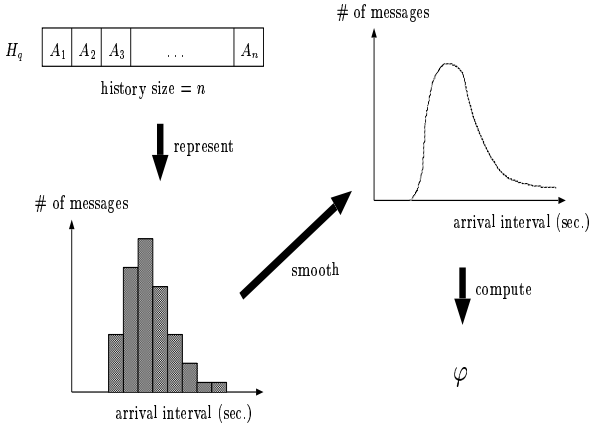

Figure 3. The mechanism for the φ-failure detector

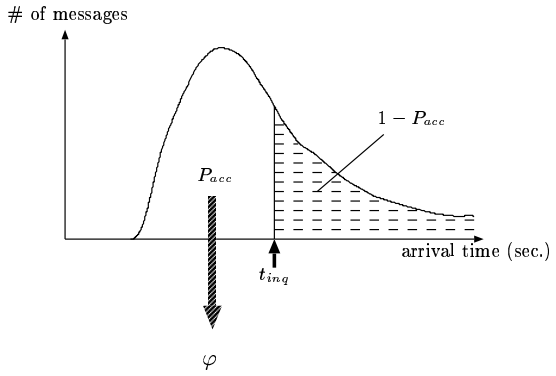

Figure 4. The translation between the distribution and φ

approach. However, they are unable to adapt to disparate application requirements. We argue that a failure detection service must be generic enough, and hence adaptive in both aspects (network conditions and application requirements) equally well.

In this paper, we have sketched the φ-failure detector approach which addresses the issue mentioned above. As far as we know, this is the first proposition for a failure detection service that can simultaneously adapt to changing network conditions and disparate application requirements, without setting practical limitations on the number of applications. In some sense, the φ-failure detector can be seen as a way to quantify the confidence of the failure detection. Our work is however still in progress and several issues must be solved before the solution can be fully implemented and used in practice. The main issues that we intend to refine and investigate in the near future are as follows.

The first issue, and arguably the most important one, is to refine the estimation of the distribution of message delays. In particular, it is important to model this distribution properly, or else the scale could become completely meaningless. Similarly, it is desirable to compute the confidence intervals associated with the estimated distribution. Indeed, this is a source of uncertainty which must be accounted for when computing φ_p.

A second issue of importance, is to consider fair-lossy channels instead of quasi-reliable ones. In other words, the failure detector must consider the possibility that heartbeat messages are lost. This might be done by estimating the loss rate of messages based on past observations. However, doing so requires some implicit assumptions that we have not identified yet.

Also, we have not discussed the case when several consecutive heartbeats are expected from process p but have not been received. The value φ_p should be computed in a way that each missing heartbeat contributes to increase the confidence that p has crashed. This would effectively ensure that every process that has crashed is eventually suspected, for any finite threshold.

Following the same idea, another issue of importance is to detail the minimal conditions (system assumptions, threshold, etc.) under which a failure detector of class $\Diamond\mathcal{P}$ can be realized.

On a different matter, the adaptation to application requirements can be significantly improved if we provide a way to convert a φ value into a failure detection time, and conversely. Indeed, different applications have different requirements. In this paper, we have focused on applications which require a certain level of accuracy. Other applications need to specify instead a guaranteed detection time. This is in fact quite difficult because a given detection time is not always enforceable. It is hence not sufficient to provide a simple conversion method; this also requires a testing a feasibility condition.

Last but not least, we intend to run practical experiments over extended periods of time and in diverse environments. We will then be able to compare our approach with traditional adaptive failure detectors. Also, we will try to measure (and possibly improve) the stability of the φ scale in real systems.

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, Sep. 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, Jun. 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] F. Cosquer, L. Rodrigues, and P. Verissimo. Using tailored failure suspectors to support distributed cooperative applications. In *Proc. of the 7th IASTED/ISMM Int'l Conference on Parallel and Distributed Computing and Systems*, pages 352–356, Washington D.C., USA, Oct. 1995.

[6] X. Défago. *Agreement-related Problems: From Semi-Passive Replication to Totally Ordered Broadcast*. PhD thesis, EPFL, Lausanne, Switzerland, 2000.

[7] X. Défago, P. Felber, and A. Schiper. Optimization techniques for replicating corba objects. In *Proc. of the 4th IEEE Int'l Workshop on Object-oriented Real-time Dependable Systems(WORDS'99)*, pages 2–8, Santa Barbara, CA, USA, Jan. 1999.

[8] X. Défago, A. Schiper, and N. Sergent. Semi-passive replication. In *Proc. 17th IEEE Intl. Symp. on Reliable Distributed Systems (SRDS-17)*, pages 43–50, West Lafayette, IN, USA, Oct. 1998.

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

[10] 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, Sep. 1999.

[11] 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.

[12] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[13] 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.

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

[15] M. Larrea, S. Arévalo, and A. Fernández. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *Proc. of the 13th Symposium on Distributed Computing (DISC'99)*, pages 34–48, Bratislava, Slovakia, 1999.

[16] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for sloving consensus. In *Proc. of the 19th Symposium on Reliable Distributed Systems (SRDS'00)*, pages 52–60, Nuremberg, Germany, 2000. IEEE Computer Society Press.

[17] A. Mostefaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In *Proc. of the Int'l Conference on Dependable Systems and Networks (DSN2003)*, San Francisco, USA, Jun. 2003.

[18] 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, Sep. 2001.

[19] 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, Jul. 1998.

[20] 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, Sep. 1998.