Performance Analysis of the φ Failure Detector

Naohiro Hayashibara and Makoto Takizawa
Dept. of Computers and Engineering
School of Science and Engineering
Tokyo Denki University (TDU)
Email:{haya,taki}@takilab.k.dendai.ac.jp

Abstract

In this paper, we explain an implementation of an accrual failure detector, that we call the φ failure detector. The particularity of the φ failure detector is that it dynamically adjusts to current network conditions the scale on which the suspicion level is expressed. We have done the experiment in a LAN in a whole day and evaluated the behavior of our φ failure detector. Then we discuss on the parameters of the failure detector based on our experimental result.

1 Introduction

It is well-known that failure detection constitutes a fundamental building block for ensuring fault tolerance in distributed systems. For this reason, many people have been advocating that failure detection should be provided as a service [7, 9, 11, 12, 14], similar to IP address lookup (DNS) or time synchronization (e.g., NTP). Unfortunately, in spite of important technical breakthroughs, this view has met little success so far. We believe that one of the main reasons is that the conventional boolean interaction (i.e., trust vs. suspect) makes it difficult to meet the requirements of several distributed applications running simultaneously. For this reason, we advocate a different abstraction that helps decoupling application requirements from issues related to the underlying system.

It is well-known that there exists an inherent tradeoff between (1) *conservative* failure detection (i.e., reducing the risk of wrongly suspecting a running process), and (2) *aggressive* failure detection (i.e., quickly detecting the occurrence of a real crash). There exists a continuum of valid choices between these two extremes, and what defines an appropriate choice is strongly related to application requirements.

One of the major obstacles to building a failure detection service is that simultaneously running distributed applications with different quality-of-service requirements must be able to tune the service to meet their own needs without interfering with each other. Furthermore, some classes of distributed applications require the use of different qualities of service of failure detection to trigger different reactions (e.g., [3, 5, 13]). For instance, an application can take precautionary measures when the confidence in a suspicion reaches a given level, and then take more drastic actions once the confidence raises above a second (much higher) level.

Failure detectors are traditionally based on a boolean interaction model wherein processes can only either trust or suspect the processes that they are monitoring. In contrast, we proposed a novel abstraction, called accrual failure detector [6], whereby a failure monitor service outputs a value on a *continuous scale* rather than information of a boolean nature.

Contribution In this paper, we measured the performance of the φ failure detector with different parameters. For this measurement we have done the experiment that a process periodically sends messages to another one in LAN during a day. Heartbeat messages were sent using UDP at a rate of about ten per second. We have got a total of nearly 750,000 samples. Using these samples, we have measured the behavior of the φ failure detector, and discuss on the impact of its parameters to its performance.

2 Failure Detectors: basic concepts & implementations

This section briefly reviews important results concerning failure detection. We first outline the basic concepts, describe important metrics, and discuss ba-

sic aspects of their implementations. At the end of the section, we describe two prior implementations of adaptive failure detectors that we later use as a reference to compare with our φ failure detector.

2.1 Unreliable failure detectors

Being able to detect the crash of other processes is a fundamental issue in distributed systems. In particular, several distributed agreement problems, such as Consensus, cannot be solved deterministically in asynchronoussystems if even a single process might crash [8]. The impossibility is based on the fact that, in such a system, a crashed process cannot be distinguished from a very slow one.

The impossibility result mentioned above no longer holds if the system is augmented with some unreliable failure detector oracle [2]. An unreliable failure detector is one that can make mistakes, to a certain degree.

2.2 Quality of service of failure detectors

Chen et al. [4] propose a set of metrics to evaluate the quality of service (QoS) of failure detectors. For simplicity and without loss of generality, they consider a simple system as follows. The system consist of only two processes called p and q, where process q monitors process p. Process p can possibly be subject to crash failures, in which case the crash is permanent. In the sequel, we consider the same system, and use the following subset of Chen's metrics.

Definition 1 (Detection time T_D) *The detection time is the time that elapses since the crash of p and until q begins to suspect p permanently.*

Definition 2 (Mistake rate λ_M) *This measures the rate at which a failure detector generates wrong suspicions.*

Notice that the first definition relates to the completeness whereas the other one relates to the accuracy of the failure detector.

2.3 Heartbeat failure detectors

In this section, we present a brief overview of heartbeat-based implementations of failure detectors. Assume that processes have also access to some local physical clock giving them the ability to measure time. These clocks may or may not be synchronized.

Using heartbeat messages is a common approach to implementing failure detectors. It works as follows (see Fig. ??): process p—i.e., the monitored process—periodically sends a heartbeat message to process q, informing q that p is still alive. The period is called the heartbeat interval Δ_i. Process q suspects process p if it fails to receive any heartbeat message from p for a period of time determined by a timeout Δ_{to}, with $\Delta_{to} \geq \Delta_i$. A third value of importance is the network transmission delay of messages. For convenience, we denote by Δ_{tr} the average transmission time experienced by messages.

In this paper, we always assume this type of failure detectors.

2.4 Adaptive failure detectors

The goal of adaptive failure detectors is to adapt to changing network conditions. Most adaptive failure detectors presented in the literature [1,4] are based on a heartbeat strategy (although nothing seems to preclude a query-response interaction style, for instance). The principal difference with using a fixed heartbeat strategy is that the timeout is modified dynamically according to network conditions.

3 Accrual Failure Detectors

The principle of accrual failure detectors is simple. Instead of outputting information of a boolean nature, accrual failure detectors output suspicion information on a continuous scale. Roughly speaking, the higher the value, the higher the chance that the monitored process has crashed. The φ failure detector has been implemented based on this abstraction.

In this section, we first describe the use of accrual failure detectors from an architectural perspective, and put this in contrast with conventional failure detectors.

3.1 Architecture overview

Conceptually, the implementation of failure detectors on the receiving side can be decomposed into three basic parts as follows.

1. *Monitoring.* The failure detector gathers information from other processes, usually through the network, such as heartbeat arrivals or query-response delays.
2. *Interpretation.* Monitoring information is used and interpreted, for instance to decide that a process should be suspected.
3. *Action.* Actions are executed as a response to triggered suspicions. This is normally done within applications.

The main difference between traditional failure detectors and accrual failure detectors is which component of the system does what part of failure detection.

In traditional timeout-based implementations of failure detectors, the monitoring and interpretation parts are combined within the failure detector. The output of the failure detector is of boolean nature; *trust* or *suspect*.

In contrast, accrual failure detectors provide a lower-level abstraction that avoids the interpretation of monitoring information. Some value is associated with each process that represents a suspicion level. This value is then left for the applications to interpret. For instance, by setting an appropriate threshold, applications can trigger suspicions and perform appropriate actions. Alternatively, applications can directly use the value output by the accrual failure detector as a parameter to their actions. Considering the example of master/worker described in the introduction, the master could decide to allocate the most urgent jobs only to worker processes with a low suspicion level.

4 Implementation of the φ accrual failure detector

In the previous section, we have presented the generic abstraction of accrual failure detectors. Accrual failure detectors can be implemented in many different ways. In this section, we present a practical implementation that we call the φ failure detector, and that we had presented in earlier work [10]. We have also shown that our failure detector can be tuned without any performance hit.

In the practical point of view, each failure detector is implemented as a daemon process in a host. The crash of the daemon is detected and then restarted by a watchdog (e.g., cron process, etc.). Thus, the crash of the failure detector means the crash of the host.

4.1 Meaning of the value φ

As mentioned, φ failure detector implements the abstraction of an accrual failure detector. The suspicion level of accrual failure detector is given by a value called φ. The basic idea of the φ failure detector is to express the value of φ on a scale that is dynamically adjusted to reflect current network conditions.

Let T_{last}, t_{now}, and $P_{later}(t)$ denote respectively: the time when the most recent heartbeat was received (T_{last}), the current time (t_{now}), and the probability that a heartbeat will arrive more than t time units after the previous one ($P_{later}(t)$). Then, the value of φ is calculated as follows.

$$\varphi(t_{now}) \stackrel{\text{def}}{=} -\log_{10}(P_{later}(t_{now} - T_{last})) \qquad (1)$$

Roughly speaking, with the above formula, φ takes the following meaning. Given some threshold Φ, and assuming that we decide to suspect p when $\varphi \geq \Phi = 1$, then the likeliness that we will make a mistake (i.e., the decision will be contradicted in the future by the reception of a late heartbeat) is about 10 %. The likeliness is about 1 % with $\Phi = 2$, 0.1 % with $\Phi = 3$, and so on. φ and Φ mean the level of suspicion. The presentation of these values are quite intuitive, because the failure detector can not decide the crash of another one with 100% confidence. However, the suspicion of the fact that it has been crashed increases over time. That's why, these values never reach to ∞ (=100%) in the suspicion level.

4.2 Calculating φ

The method used for estimating φ is in fact rather simple. This is done in three phases. First, heartbeat arrive and their arrival times are stored in a sampling window. Second, these past samples are used to determine the distribution of inter-arrival times. Third, the distribution is in turn used to compute the current value of φ.

4.2.1 Sampling heartbeat arrivals

The monitored process (p in our model) adds a sequence number to each heartbeat message. The monitoring process (q in our model) stores heartbeat arrival times into a sampling window of fixed size WS. Whenever a new heartbeat arrives, its arrival time is stored into the window, and the data regarding the oldest heartbeat is deleted from the window. Arrival intervals are easily computed. In addition, to constantly determine the mean μ and the variance σ^2, two other variables are used to keep track of the sum and sum of squares of all samples in the window.

4.2.2 Estimating the distribution and computing φ

The estimation of the distribution of inter-arrival times assumes that inter-arrivals follow a normal distribution. The parameter of the distribution are estimated from the sampling window, by determining the mean μ and the variance σ^2 of the samples. Then, the probability $P_{later}(t)$ that a given heartbeat will arrive more than t time units later than the previous heartbeat is given by the following formula.[1]

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_{t}^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \mathrm{d}x \qquad (2)$$

$$= 1 - F(t) \qquad (3)$$

[1] The formula is simplified assuming that crashes are rare events.

where $F(t)$ is the cumulative distribution function of a normal distribution with mean μ and variance σ^2.

Then, the value of φ at time t_{now} is computed by applying Equation 1 described in Section 4.1.

5 Experimental Results

In this section, we study the behavior of the φ failure detector when used over a wide-area network. The measurements have been taken in a rather extreme environment (wide area network, short heartbeat interval) to assess both the robustness and the scope of applicability of the failure detector.

First, we describe the environment in which the experiments have been conducted. Second, we study the effect of several parameters on the behavior of the φ failure detector. Third, we compare the results obtained using the φ failure detector with that of Chen and Bertier (see §2.4).

5.1 Environment

Our experiments involved two computers running on Linux OS in the local area network (LAN) at Tokyo Denki University. One machine was running program sending heartbeats (thus acting like process p) while the other one was recording the arrival times of each heartbeat (thus acting like process q). Neither machine failed during the experiment. Our experiments have been done on the two different environments in the LAN. We summarize the characteristic of these environments in Fig. 1.

5.2 Setting and result

To conduct the experiments, we have recorded heartbeat sending and arrival times using two nodes on the different environments. Then, we replayed the receiving times recorded for the φ failure detector and every different value of the parameters. As a result, all settings were compared based on *exactly* the same scenarios, thus resulting in a fair comparison.

All experiments we conducted rely on a window of past samples to compute their estimations. Unless stated otherwise, the failure detectors were set using different window sizes; 20, 100, 500, 2,000 and 10,000 samples. For each window size, we have tuned the threshold Φ from 0.5 to 16.0.

Figure 2 and 3 shows the results obtained when plotting the mistake rate on a logarithmic scale. The mistake rate means the rate at which a failure detector generates wrong suspicions in an unit of time.

Figure 2 shows that smaller window size only have aggressive range (i.e., early detection time) but it can have better performance than longer ones in such a range. For instance, the performance of the failure detector with 20 samples (i.e., window size (WS) = 20) is better than one with 2,000 samples. While, the failure detector with 20 samples needs larger Φ than one with 2,000 samples to have lower mistake rate at the same detection time.

While the failure detector with large window size (e.g., WS = 2,000) allows to have conservative failure detection. Such a tuning of the parameters also realizes extremely high accuracy on the failure detection but it takes very long time to detect a failure (Fig.2).

In the other hand, Fig. 3 shows almost the same curves but longer window size allow to make high precision on the failure detection in the range of early detection time $T_D \in [0.5; 2.0]$ (sec.).

5.3 Discussion on parameters

The φ failure detector allow users or applications to set the window size and threshold Φ for each target node. Thus, these parameters fix the quality of the failure detection for the target. In Fig. 2 and Fig. 3, each setting (i.e., window size and threshold) corresponds to each point.

If a point is getting close to the bottom left corner (the corner with zero in both axises), it means that the performance of the failure detector improves. While, if it moves to the upper right corner, the failure detector reduces its performance.

Now we consider that the φ failure detector satisfies the QoS requirement by tuning its parameters. We might have several choices on the combination of the threshold Φ and window size.

According to our experimental result shown in Fig. 3, you need a longer window size and a large threshold (e.g., WS = 2,000 and Φ = 13.0) if you require high accuracy. In this environment, this setting also leads reasonably early detection time. However, it is not always make such a result. A smaller window size (e.g., WS $\leq$ 100) makes better result than a larger one in Fig. 2. It depends on a distribution of heartbeat arrival time. The difference between two environments is the standard deviation of arrival times. The environment of Fig. 2 has a large value in the standard deviation of heartbeat arrival times than the one of Fig. 3 (See Fig. 1). The setting with smaller window size (e.g., 20 $\leq$ WS $\leq$ 100) would be useful to cover the left-upper corner.

6 Conclusion

We have presented the performance measurements of the φ failure detector, an instance of the more general abstraction of accrual failure detectors. These

Title	avg. inter-arrival time [sec.]	avg. throughput [10^6 bits/sec.]	ratio of message loss
Env. 1	0.117 (5.00)	7.40 (2.31)	0.014
Env. 2	0.101 (0.04)	10.28 (0.80)	0.016

Figure 1. Environments for the performance measurements. A numerical number in a bracket meas the standard deviation.

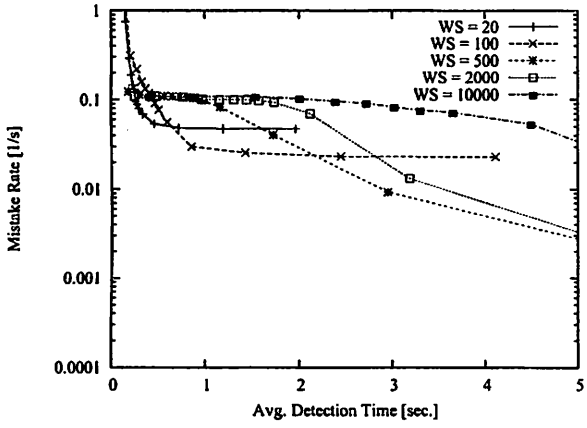

Figure 2. Performance measurement of the φ failure detector with different window size in Env. 1 Each point corresponds to the threshold $\Phi \in [0.5; 13.0]$.

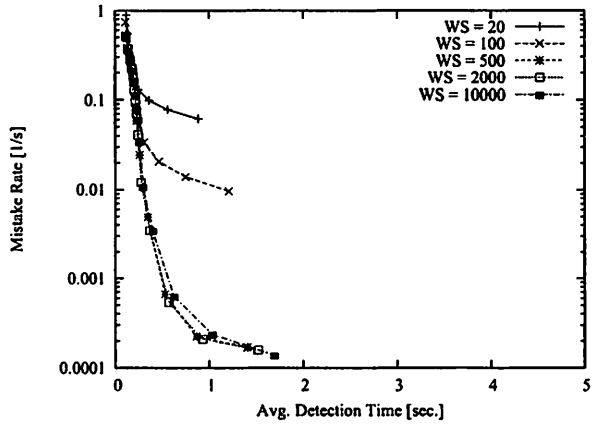

Figure 3. Performance measurement of the φ failure detector with different window size in Env. 2 Each point corresponds to the threshold $\Phi \in [0.5; 13.0]$.

measurements have been done on the two different environments mentioned above. We have discussed the tuning of the parameters of the φ failure detector to have QoS requirements.

In addition to interesting observations about window size, our experimental results show that our failure detector can behave with high accuracy in aggressive range even with small window size (e.g., window size = 20). The failure detector with small window can be available quickly from its bootstrapping. We found the impact of window size. Specially, small window leads to low mistake rate at early detection time. While, our failure detector with long window overcomes one with small window on accuracy of failure detection in the conservative range (long detection time).

We are currently working on performance measurements of the φ failure detector in various environments. We also develop our failure detector to improve its performance.

Acknowledgement

We are thankful to Xavier Défago for his collaboration on accrual failure detectors. This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists B (Nr. 17700055).

References

[1] 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.

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

[3] B. Charron-Bost, X. Défago, and A. Schiper. Broadcasting messages in fault-tolerant distributed systems: the benefit of handling input-triggered and output-triggered suspicions differently. In *Proc. of the 21st IEEE Int'l Symposium on Reliable Distributed Systems (SRDS-21)*, pages 244–249, Osaka, Japan, Oct. 2002.

[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, 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.

[6] 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)*, pages 206–215, Yokohama, Japan, June 2005.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] P. Urbán, I. Schnayderman, and A. Schiper. Comparison of failure detectors and group membership: Performa nce study of two atomic broadcast algorithms. In *Proc. Intl. Conf. on Dependable Systems and Networks (DNS'03)*, pages 645–654, San Francisco, CA, USA, June 2003.

[14] 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.