A Class of Practical Self-tuning Failure Detection Schemes for Distributed Networks

Naixue Xiong
Dept. of Computer Science
Georgia State Univ., USA
E-mail: nxiong@cs.gsu.edu

Athanasios V. Vasilakos
Dept. of Comp. and Tele. Engi.
Univ. of Western Macedonia, Greece
E-mail: vasilako@ath.forthnet.gr

Jie Wu
Dept. of Comp. and Info. Scie.
Temple Univ., USA.
E-mail: jiewu@temple.edu

Y. Richard Yang
Dept. of Computer Science
Yale Univ., USA
E-mail: yry@cs.yale.edu

Andy Rindos
IBM Corporation, Dept. W4DA/Bldg 503
Research Triangle Park, NC, USA
E-mail: rindos@us.ibm.com

Yi Pan
Dept. of Computer Science
Georgia State Univ., USA
E-mail: pan@cs.gsu.edu

***Abstract*—Cloud computing is an increasingly important solution for providing services deployed in dynamically scalable cloud networks. Services in the cloud computing networks may be virtualized with specific servers which host abstracted details. Some of the servers are active and available, while others are busy or heavy loaded, and the remaining are offline for various reasons. Users would expect the right and available servers to complete their application requirements. Therefore, in order to provide an effective control scheme with parameter guidance for cloud resource services, failure detection is essential to meet users' service expectations. It can resolve possible performance bottlenecks in providing the virtual service for the cloud computing networks. Most existing Failure Detector (FD) schemes do not automatically adjust their detection service parameters for the dynamic network conditions, thus they couldn't be used for actual application. This paper explores FD properties with relation to the actual and automatic fault-tolerant cloud computing networks, and find a general non-manual analysis method to self-tune the corresponding parameters to satisfy user requirements. Based on this general automatic method, we propose a specific and dynamic Self-tuning Failure Detector, called SFD, as a major breakthrough in the existing schemes. We carry out actual and extensive experiments to compare the quality of service performance between the SFD and several other existing FDs. Our experimental results demonstrate that our scheme can automatically adjust SFD control parameters to obtain corresponding services and satisfy user requirements, while maintaining good performance. Such an SFD can be extensively applied to industrial and commercial usage, and it can also significantly benefit the cloud computing networks.**

***Index Terms*—Application requirements, Cloud computing networks, Fault tolerance, Quality of service, Self-tuning failure detection**

I. Introduction

Cloud computing provides resources to satisfy large numbers of user applications across networks [1-3]. Cloud computing networks [4-5] link users across the globe. For instance, they are being developed to support an education infrastructure for student courses, learning labs, or connecting teachers and students to expert mentors [6]. Cloud models should maintain high levels of Quality of Service (QoS) in accessing information remotely in network environments, and should ensure users' application security and dependability.

In the cloud computing networks, some of the servers may be active and available, while others are busy or heavy loaded, and the remaining may be offline or even crashed for various reasons. Therefore, the cloud service environment can be dynamic and unexpected [7], and users would expect the right and available servers to complete their application requirements. We must be able to address the variability and provide an effective control scheme with parameters guidance to guide service conditions and cloud resources. Thus fault-tolerant schemes are designed to provide reliable and continuous services in cloud computing networks despite the failures of some of their components [8-19]. As an essential building block for the cloud computing networks, a Failure Detector (FD) plays a critical role in the engineering of such dependable network systems [19]. Effective failure detection is essential to ensure acceptable QoS, and it is necessary to find an optimized FD that can detect failures in a timely and accurate way before a generic FD service can actually be implemented for cloud computing network applications [20]. For example, PlanetLab is a global cloud computing network that supports the development of new network services and it currently consists of 1076 nodes at 494 sites. While lots of nodes are inactive at any time, yet we do not know the exact status (active, slow, offline, or dead). Therefore, it is impractical to login one by one without any guidance.

The design of reliable FDs is a very difficult task. One of the main reasons is that the statistical behavior of communication delays is unpredictable. The other reason is that asynchronous (i.e., no bound on the process execution speed or message-passing delay) distributed systems make it impossible to determine precisely whether a remote process is failed or has just been very slow [21]. An unreliable FD [21] can make mistakes like falsely suspecting correct processes or trusting crashed processes. To ensure acceptable QoS for such an unreliable FD, parameters should be properly tuned to deliver a desirable QoS at the upper layers, because the QoS of an FD greatly influences the QoS that upper layers may provide. Many fault-tolerant algorithms have been proposed (e.g., [19, 22-27]), which are never-the-less based on unreliable FDs.

A set of metrics are proposed to quantify the QoS of an FD by Chen et al. in [28]: how fast it detects actual failures and how well it avoids false detections. In order to improve the QoS of an FD, a lot of adaptive FDs have been proposed [29-32], such as Chen FD [28], Bertier FD [29, 33], and the ϕ FD [30-31]. Chen et al. in [28] proposed several implementations relying on the probabilistic behavior of the network systems. The protocol uses arrival times sampled in the recent past to compute an estimation of the arrival time of the next heartbeat. However, a timeout that is set according to this estimation plus a constant safety margin, does not match the dynamic network behavior well [29]. Subsequently, Bertier FD [29, 33] provides an optimization of the safety margin for Chen FD. It uses a different estimation function, combining Chen's and Jacobson's estimation of the Round-Trip Time (RTT). Bertier FD is primarily designed to be used over wired local area networks (LANs), where messages are seldom lost [30]. The self-tuned FDs in [34-35] use the statistics of the previously-observed communication delays to continuously adjust timeouts. In other words, they assume a weak past dependence on communication history. These three FDs dynamically predict new timeout values based on observed communication delays to improve the performance of the protocols.

Even though the above FDs had important technical breakthroughs, their success was limited. As far as we know, there are three main reasons [30]: (1) an FD provides an information list of suspects about which processes have crashed. This information list is not always up-to-date or correct (e.g., an FD may falsely suspect a process that is alive), due to the high unpredictability of message delays, the dynamic and changing topology of a network system, and the high probability of network message losses; (2) The conventional binary interaction (i.e., trust and suspect) makes it difficult to satisfy the requirements of several distributed applications running simultaneously. In practice, many classes of distributed network applications require the use of different QoS of failure detection to trigger different reactions (e.g., [36-37]). For instance, an application may take precautionary network measures when the confidence in a suspicion reaches a given low level, while it takes successively more drastic actions once the doubt progresses to higher levels [10]. However, the traditional output of the FDs (Chen FD [28] and Bertier FD [29, 33]) is of a binary nature[1]. (3) Most existing schemes can not automatically adjust their parameters for dynamic network conditions, though users with an awareness of the scheme's internal core functions and parameters can provide suitable values for the parameters based on the output QoS, which should satisfy the users' expectation of QoS ($\overline{QoS}$). This tuning may be very complex, and it is best done by professional engineers. Essentially, we are unable to use fixed parameters for FDs to satisfy user requirements in a reliable way, where networks are dynamic and unexpected. In sum, the parameters used by existing failure detectors require adjustment by hand, and so are not suitable for dynamic networks, especially in large scale distributed networks or unstable networks[2]. Therefore, we seek to further explore the FD properties and their inter-relations in providing an actual fault-tolerant distributed system.

To attack the above problems, this paper firstly proposes a general self-tuning failure detection method in the fault-tolerant cloud computing networks, and it can be extensively used for industrial and commercial purposes. *In contrast, other schemes could blindly provide different output QoS*: some are just temporarily suitable for the requirements; yet some are never, then engineers have to manually change the relevant parameters. *These schemes must try all the possible parameter values, and get a performance output graph to know which parameter values are acceptable for the network (manually choose relevant parameters). If the network has significant changes, the engineers have to change the relevant parameters manually again.* Secondly, based on the above general method, we propose an automatic Self-tuning Failure Detector (SFD) to optimize the existing FDs, as an accural FD[3]. Thirdly, we explore the implementation of SFD, which, briefly speaking, works as follows: A sliding window maintains the most recent samples of the arrival time, similar to conventional adaptive FDs [10, 28-29]. The next timeout delay τ approximation (for next sample) is adjusted by the sliding window and feedback information from the output. With this dynamic feedback information, the relevant parameters are computed to match recent network conditions. By design, SFD can adjust well to handle unexpected network conditions and the requirements of any number of concurrently running applications. Finally, we comparatively evaluate our failure detection scheme and existing schemes (Chen FD [28], Bertier FD [29, 33], and ϕ FD [30-31]) by extensive experiments in seven representative Wide Area Network (WAN) cases. The experimental results show the properties of the different FDs, and demonstrate that our scheme can automatically adjust SFD control parameters to obtain corresponding services and satisfy user requirements, while maintaining good performance. Such an SFD can be extensively applied to industrial and commercial usage, and it can also significantly benefit the cloud networks.

The remaining content of the paper is organized as follows: In Section II, the system model and failure detection QoS metrics are introduced. Section III introduces several adaptive FDs. In Section IV, we present a general self-tuning failure detection method for applicable engineering of such fault-tolerant cloud computing networks. Based on the general method, we propose a SFD to optimize the existing FDs, and explore the implementation of SFD. Section V carries out experiments in different network conditions (seven representative WAN cases). In Section VI, we discuss more work related to FDs. Finally, we conclude our work and discuss further work in Section VII.

II. System Model and Basic Concepts

In this section, we first propose the practical cloud computing network model (see Fig. 1) and academic cloud computing network analysis model (see Fig. 2), and then define the failure detection QoS metrics.

A. *Practical Cloud Computing Network Model*

As shown in Fig. 1, we explore a dynamic cloud computing network model[4], which is based upon the United States south-

[1]Bertier FD and Chen FD were aimed at other problems, which they both solved admirably well.

[2]Here it means the networks have the high unpredictability of message delays, the great dynamic changing topology of system, or the high probability of message losses.

[3]Accrual FD is when an FD service outputs a suspicion level on a continuous scale rather than information of a boolean nature (trust vs suspect).

[4]In particular, we note that the general model can include multiple clouds.

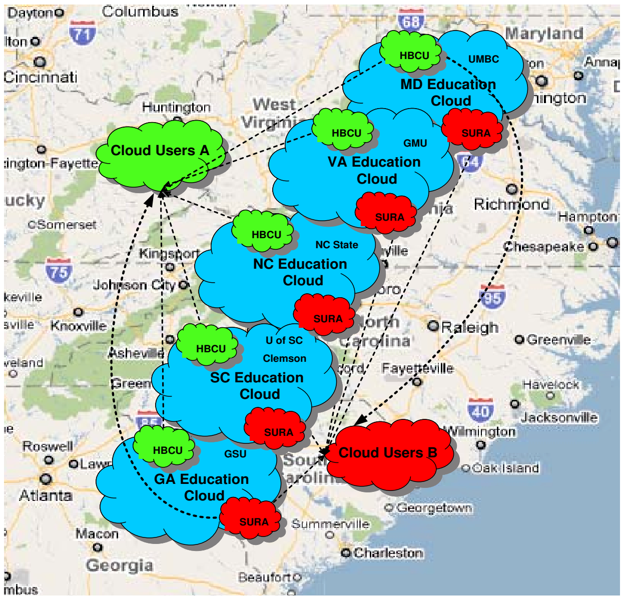

Fig. 1. Dynamic cloud computing network model: U.S. southern state education cloud consortium (sponsored by IBM, SURA & TTP/ELC).

ern states education cloud consortium and a network structure for data centers [7]. Currently, five southern states (Georgia (GA), South Carolina (SC), North Carolina (NC), Virginia (VA), Maryland (MD)) have education cloud initiatives underway (Fig. 1). The multiple clouds could provide services to each other, such as depicted with users from the African-American Colleges and Universities (HBCU) community and users from the Southeastern Universities Research Association (SURA), who may access the others' resource (the curved dotted lines in Fig. 1). Guo et al. [7] has presented a similar fault-tolerant network structure for data centers, called DCell, where a DCell with a small server node degree can support up to several million servers, just like the "Education cloud" in Fig. 1. Thus, Fig. 1 is an actual and representative cloud computing network model.

As mentioned above, servers' statuses may vary in cloud computing networks: some are active, some busy or very slow, and some may be dead or crashed. Thus, a cloud failure detection model is important to address failures of expected services by the corresponding servers, and initiate measures to ensure appropriate service response. There are several existing FD schemes that could provide some detection services, while not by automatically adjusting their parameters for dynamic network conditions. Therefore, this paper presents a general self-tuning method to solve the above problem, and also gives an exact example based on the general method and the above model.

B. *A Theoretical Cloud Computing Network Model*

Here we consider a partially synchronous cloud computing network system consisting of a finite set of processes $\Pi = \{p_1, p_2, p_3, \ldots, p_n\}$[5]. A process may fail by crashing, here a crashed process does not recover. A process behaves correctly (i.e., according to the specification) until it (possibly) crashes. By definition, a correct process is a process that does not crash, and a faulty process is a process that is not correct.

It is assumed that every pair of processes is assumed to be connected by one unidirectional unreliable communication channel [17]. An unreliable channel is defined as a communication channel: there is no message creation, no message alteration and no message duplication, while it is possible to lose some messages. Processes are completely connected via unidirectional unreliable communication channels. Without loss of generality, this paper considers a simple system model (same as [28-31, 33]) with only two processes, called p and q, which are arbitrarily taken from the large system Π, where process q monitors process p (see Fig. 2): p may periodically send a message to q, perform local computation, or is subject to crash[6]. Here the sending period is called the heartbeat interval Δt. Process q may receive a message from p, or perform local computation. q suspects process p if it can't receive any heartbeat message from p for a period of time determined by the freshpoint (FP), which is given by the parameter (timeout τ).

As illustrated in Fig. 2, d_i is the transmission delay of heartbeat m_i from p to q, and the sending period is called the heartbeat interval Δt. For the incoming heartbeat m_i, q dynamically gives a response based on the new freshpoint FP_i. This model describes four cases that may occur. The first one is that heartbeat message m_1, from the sending time σ_1 of process p, arrives at q before q's freshpoint FP_1, then q trusts p from the m_1 arrival time (here we assume that p is trusted in the initial case). The second case is that the heartbeat message m_2 from p is lost, then q waits for that heartbeat until its freshpoint FP_2; after that, q starts to suspect p. The third case is that heartbeat m_i from p arrives at q after q's freshpoint FP_i, then q suspects p from FP_i until the m_i arrival time. In the fourth case, after p sends out the heartbeat $m_{(i+1)}$, p is crashed. For the incoming heartbeat $m_{(i+1)}$, q computes a new freshpoint FP_{i+1} based on different FD schemes, and then gives a response based on the heartbeat arrival time.

Our SFD assumes the existence of some global time (unknown to processes) denoted by global stabilized time, and that processes always make progress. Furthermore, at least $\delta > 0$ time units elapse between consecutive steps (the purpose of the latter is to exclude the case where processes take an infinite number of steps in finite time) [31]. The inter-process communication model is based on message exchanges over the User Datagram Protocol (UDP) communication protocol. We don't consider the relative speed of processes; however, we consider that processes have access to a local clock device used to measure the passage of time. Furthermore, every process has access to a failure detection service.

It is commonly believed that the period Δt is a factor that contributes to the detection time. However, Müller [38] indicates that Δt is little determined by QoS requirements on several different networks, but much by the characteristics of the underlying system, and the work in [30] suggests that there exists some nominal range for the parameter Δt with little or no impact on the accuracy of the FD in every network.

[5]This model is based on the above IBM cloud computing networks in Fig. 1. A user, a manager, or a total education cloud is regarded as a process.

[6]Based on the theoretical cloud model in Fig. 1, here process q is like a manager, and process p is like an education cloud. Then every education cloud service environment is given by the monitoring results.

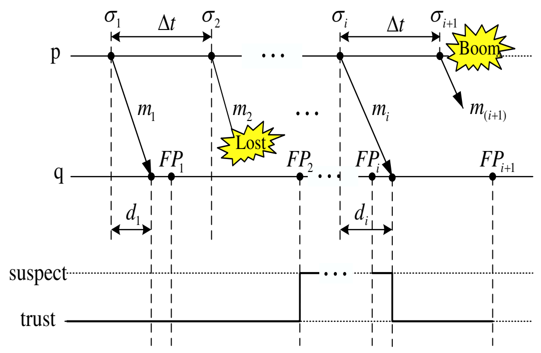

Fig. 2. Basic heartbeat failure detection model.

The freshpoint is fixed in the conventional implementation of this model. If the time between two next freshpoints is too short, the likelihood of a wrong suspicion rate is high, though crashes are detected quickly. In contrast, if the time is too long, there will be too much detection time, although there are fewer inaccurate suspicions.

Alternatively, this model can set the freshpoint based on the transmission delay of the heartbeat. The advantage is that the maximal detection time is bounded, but the disadvantage is that it relies on physical clocks with a negligible drift[7] and a shared knowledge of the heartbeat interval Δt. The drawback is a serious problem in practice: when the regularity of the sending of heartbeats cannot be guaranteed, and the actual sending interval is different from the target one (e.g., timing inaccuracies due to irregular OS scheduling) [30]. Both of the methods have their respective advantages and disadvantages, and it is difficult to conclude which one is better [30].

C. Failure detection QoS metrics

To quantitatively evaluate the QoS of FDs, we use three main QoS metrics (i.e., detection time, mistake rate, and query accuracy probability) that are independent [28]. The first metric measures the impact on the model from the speed of the FD, and the other two metrics relate to accuracy. In detail, considering two processes p and q where q monitors p, the QoS of the FD at q (called fd_q) can be determined from its transitions between the "trust" and "suspect" states with respect to p (see Fig. 3[8]).

Detection Time (T_D): This is a random variable that represents the length of a period from the time when p starts crashing to the time when q starts suspecting p permanently by fd_q. **Mistake Rate** (MR): This is a random variable that represents the number of mistakes that the failure detector makes in a unit of time, i.e,. it represents how frequently the failure detector makes mistakes. **Query Accuracy Probability** (QAP): This is a probability that, when queried at a random time, the FD at q indicates correctly that process p is up [28].

Failure Detection QoS Definition Based on the work in [39], a particular FD performance is defined in terms of its completeness and accuracy properties, and the QoS provided by each of its constituent failure detection modules is a tuple [39]:

$$QoS = (T_D, MR, QAP). \tag{1}$$

The QoS quantifies both how fast a detector suspects a failure and how well it avoids false detection.

[7]A straightforward implementation of clocks requires synchronized clocks. Chen et al. [28] shows the method to do it with unsynchronized clocks, but this still requires the drift between clocks to be negligible.

[8]In this Fig. 3, T_M (Mistake duration) measures the time that elapses from the beginning of a wrong suspicion until its end (i.e., until the mistake is corrected). T_{MR} (Mistake recurrence time) measures the time between two consecutive wrong suspicions, it is a random variable representing the time that elapses from the beginning of a wrong suspicion to the next one.

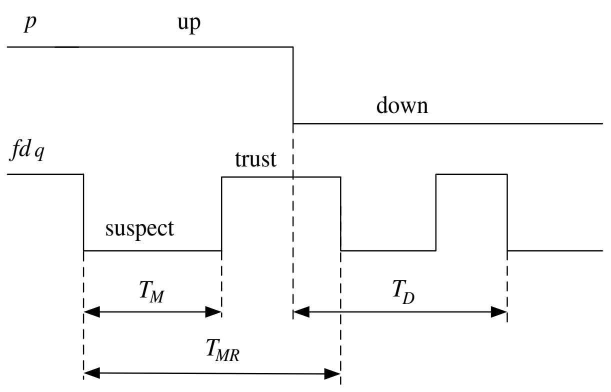

Fig. 3. Basic Metrics for the QoS evaluation of an FD [28].

III. Existing Adaptive failure detectors

Recently, research studies have focused on the adaptive FDs, whose goal is to adapt to changing network conditions and application requirements [30]. In general, most adaptive FDs are based on a heartbeat strategy, where recent historical information is used to predict the arrival time of the next heartbeat. Existing main adaptive FDs (Chen FD [28], Bertier FD [29, 33], and ϕ FD [30-31]) work as follows:

Chen FD Chen et al. [28] assumes that process p sends heartbeat messages periodically to process q (see Fig. 2). The most recent n heartbeat messages in a sliding window, denoted by m_1, m_2, ..., m_n, are considered by process q. A_1, A_2, ..., A_n are their actual receiving times according to q's local clock. When at least n messages have been received, the theoretical arrival time $EA_{(k+1)}$ can be estimated by:

$$EA_{(k+1)} = \frac{1}{n} \sum_{i=k-n-1}^{k} (A_i - \Delta_i * i) + (k+1)\Delta_i, \tag{2}$$

where Δ_i is the sending interval. The next timeout delay (which expires at the next freshness point $\tau_{(k+1)}$) is composed of $EA_{(k+1)}$ and constant safety margin α. One has

$$\tau_{(k+1)} = \alpha + EA_{(k+1)}. \tag{3}$$

This technique provides an estimation for the next arrival time based on a constant safety margin.

Bertier FD Bertier et al. [29, 33] estimated the safety margin is dynamically based on Jacobson's estimation of the RTT [39]. Bertier FD adapts the safety margin α based on the variable $error$ in the last estimation. The recursive algorithm [29, 33] is as follows:

$$error_k = A_k - EA_{(k)} - delay_{(k)}, \tag{4}$$

$$delay_{(k+1)} = delay_{(k)} + \gamma \cdot error_{(k)}, \tag{5}$$

$$var_{(k+1)} = var_{(k)} + \gamma \cdot (|error_{(k)}| - var_{(k)}), \tag{6}$$

$$\alpha_{(k+1)} = \beta \cdot delay_{(k+1)} + \phi \cdot var_{(k)}, \tag{7}$$

and

$$\tau_{(k+1)} = EA_{(k+1)} + \alpha_{(k+1)}, \tag{8}$$

where the parameter γ represents the importance of the new measure with respect to the previous ones, the variable $delay$

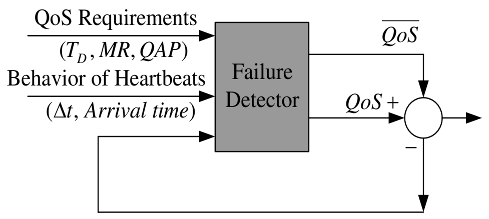

Fig. 4. Feedback architecture in self-tuning failure detector with the target $\overline{MR}$ and $\overline{T_D}$.

represents the estimate margin, and var estimates the magnitude of errors. β and ϕ are used to adjust the variance var. Typical values of β, ϕ and γ are 1, 4 and 0.1, respectively. Bertier's estimation provides a short detection time.

ϕ **FD** Different from the above schemes, the ϕ FD [30-31] outputs a suspicion level on a continuous scale, instead of providing information of a conventional binary nature (trust or suspect). In this scheme, T_{last} denotes the time when the most recent heartbeat was received; t_{now} is the current time; and $P_{later}(t)$ denotes the probability that a heartbeat will arrive more than t time units after the previous one. Then, the value of ϕ is calculated as follows:

$$\phi(t_{now}) = -lg(P_{later}(t_{now} - T_{last})). \tag{9}$$

Here,

$$P_{later}(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_t^{+\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}} dx = 1 - F(t), \tag{10}$$

where $F(t)$ is the cumulative distribution function of a normal distribution with mean μ and variance σ^2, and μ and σ^2 parameters of the distribution are estimated from the sampling window, where inter-arrival times are saved. The value of ϕ at time t_{now} is computed by applying the Equation (9).

When the value of ϕ is returned to the applications, every application compares the value of ϕ with its threshold Φ, which is given by users based on different applications. If $\phi > \Phi$, a certain action is triggered. Therefore, for different applications with various Φ, corresponding explanations are provided; for the same application, a different value of ϕ can trigger various actions. Thus, in ϕ FD, Φ should be careful to be chosen to achieve good performance in an actual engineering application systems.

IV. A General Self-tuning Failure Detection Scheme

Here we first present a general self-tuning failure detection method for the applicable engineering of such fault-tolerant cloud computing networks. Then SFD, an optimization of the existing FDs, is presented as an effective example. Finally, the implementation of SFD is described precisely.

A. A General Self-tuning Failure Detection Method

Based on the system mode in Fig. 2, the user p hopes FD in process q detects p with a certain QoS requirement. In addition, the SFD in q can adjust its parameters by itself to satisfy the $\overline{QoS}$.

In Fig. 4, we show the feedback architecture in SFD, where $\overline{QoS}$ is the target QoS for the heartbeats. The initial QoS requirements (T_D, MR, QAP) and $\overline{QoS}$ are known and sent to the SFD, and the network behaviors (for example, the heartbeat information: arrival time, heartbeat sending inter-arrival time Δt) are also sent into the SFD. Combining the feedback information from the output, the SFD could adjust its parameters to match the target $\overline{QoS}$ requirement.

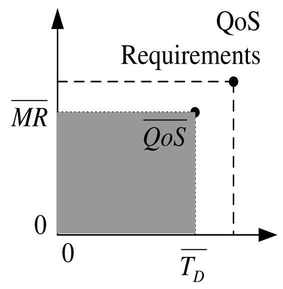

Fig. 5. Parameter relation of self-tuning failure detection, $\overline{QoS}$ is the target QoS for the heartbeats.

If the output QoS of the SFD does not satisfy the target $\overline{QoS}$ (for example, we could define it as $QoS > \overline{QoS}$), then the feedback information ($QoS - \overline{QoS}$) is returned to SFD. Based on the feedback information, SFD adjusts its parameters (for example, timeout τ for the timeout-based schemes). Then, eventually SFD can satisfy the $\overline{QoS}$ (if there is a certain range for this SFD, where SFD can satisfy the $\overline{QoS}$). Otherwise, if the $\overline{QoS}$ is too high, and this SFD could not find suitable parameters for it, then the SFD will give a response: "This SFD can not satisfy the $\overline{QoS}$ for the application".

For more details, if we focus on the three main parameters in QoS: T_D, MR, and QAP (the performance parameters for a period experiment, not for a time slot), then the output QoS of SFD is based on all the former time periods.

In Fig. 5, we show the parameter relation of self-tuning failure detection, where the target $\overline{MR}$ and $\overline{T_D}$ should be smaller than the required values of MR and T_D, and the $\overline{QAP}$ should be larger than the required values of QAP.

In effect, in a specific time slot, we adjust the parameters of SFD only one time, based on feedback information, to improve the output QoS of SFD to close the $\overline{QoS}$. Usually we have to repeatedly adjust the parameters of SFD in multiple time slots to improve the output QoS gradually, and finally find the proper parameters to satisfy the $\overline{QoS}$. At that time, the SFD stabilizes the parameters of SFD and cloud communication network systems. If systems have great changes and the responding output QoS does not satisfy the $\overline{QoS}$, then the SFD will give feedback information to improve output QoS of SFD gradually again until the output QoS of SFD satisfies the $\overline{QoS}$. Here we assume the experimental time is long enough to let output QoS of SFD satisfy the $\overline{QoS}$ for the applications and the proper control parameters are existing and available. This method is general, and can be applied to the other adaptive timeout-based FD schemes (see Algorithm 1). For Algorithm 1, we should first get the output QoS (T_D, MR, and QAP) based on the traditional detection scheme [28], and then try to get the feedback information to adjust the relevant parameters in SFD.

B. Self-tuning Failure Detector

Based on the above general self-tuning failure detection method, here we present a material SFD for the engineering application, which also optimizes the existing failure detectors as an example.

Here, we combine the Chen FD [28] and ϕ FD [30-31] schemes. Because Chen FD [28] has an extensive performance range, it could achieve better performance in a conservative range than ϕ FD and Bertier FD [29, 33], and also achieve similar performance to ϕ FD in an aggressive range. ϕ FD is available in only the aggressive range because its rounding errors prevent to compute points in the conservative range. Bertier FD has no dynamic parameter, and has only one aggressive performance value. Furthermore, ϕ FD outputs a suspicion level on a continuous scale (not traditional binary information), and could provide different QoS of failure detection to trigger different reactions.

Algorithm 1 A General Self-tuning Failure Detection Method

```
1:  Begin
2:  Initialization:
3:      $\overline{QoS}$: Set the target QoS;
4:      Define a self-tuning FD and its basic and initial parameters;
5:  Steps: Self-tuning Failure Detection
6:      Get the output QoS ($T_D$, MR, QAP);
7:      Get the feedback information $(QoS - \overline{QoS})$;
8:      If the feedback information is an alarm response (this SFD
        can not get this high QoS requirement): show this response to
        users and stop SFD (go to line 10 in this algorithm);
9:      Adjust the relevant parameters in SFD if the output QoS does
        not satisfy $\overline{QoS}$; otherwise, fix the parameters.
10: End
```

SFD adjusts the next predictable freshness point $\tau_{(k+1)}$ based on the feedback information. Therefore, we have

$$\tau_{(k+1)} = SM_{(k+1)} + EA_{(k+1)}, \tag{11}$$

where $EA_{(k+1)}$ is the same as the parameter in Chen-FD, while SM is the dynamic safety margin, and can be adjusted to satisfy the predefined $\overline{QoS}$. Here, we have

$$SM_{(k+1)} = SM_k + Sat_k\{QoS, \overline{QoS}\} \cdot \alpha, \tag{12}$$

where the α ($\alpha \in (0, 1)$) is the same as the constant safety margin in Chen-FD, and we set

$$Sat_k\{QoS, \overline{QoS}\} = \begin{cases} \pm\beta, QoS > \overline{QoS}; \\ 0, QoS \leq \overline{QoS}. \end{cases} \tag{13}$$

where β is a constant value, and $\beta \in (0, 1)$, and based on the specific output QoS status, $Sat_k\{QoS, \overline{QoS}\}$ could be set as β, $-\beta$, or 0. The value β is for the adjusting rate, and it could be dynamically chosen by users.

From Functions (11-13), a larger α value will lead to larger T_D, shorter MR, and larger QAP in our SFD (because a larger α value provides a larger safety margin). To this point, our scheme is similar to Chen-FD. To choose the $Sat_k\{QoS, \overline{QoS}\}$, we focus on the two aspects: response time (T_D) and detection precision (MR and QAP) (see Algorithm 2). We should make a compromise between response time and detection precision to match the target QoS $\overline{QoS}$. For example, if we try to shorten response time, and then this adjusting will worsen the detection precision, and vice versa.

From a theoretical view, SFD satisfies the property of the accrual failure detector [31], and also belongs to the class $\Diamond\mathcal{P}_{ac}$ (accruement property and upper bound property), which is sufficient to solve the consensus problem.

Theorem 1: SFD implements an FD of class $\Diamond\mathcal{P}_{ac}$, on the condition that the system is in accordance with the system model defined in Section II (see the proof of Theorem 1 in the Appendix).

C. The Implementation of SFD

This section first describes the architecture of SFD, then presents its specific implementation algorithm.

1) The Architecture of SFD: Conceptually, the implementation of SFD can be decomposed into three basic parts: Monitoring, Interpretation, and Action [30].

In traditional timeout-based FDs (Chen FD [28] and Bertier FD [29, 33]), the monitoring and the interpretation are combined within the FD, and the output is binary. However, SFD, as an accrual FD [30-31], provides a lower-level abstraction that avoids the interpretation of monitoring information. Some values, the suspicion level associated with each process, are left for the applications to interpret [30].

Application processes set a suspicion threshold according to their own QoS requirements: a low threshold generates many wrong suspicions, but quickly detects an actual crash. Conversely, a high threshold is prone to generate fewer mistakes, but needs more time to detect actual crashes.

2) The Implementation of SFD: As an accrual FD, the method used in SFD is quite simple. After a warm-up period, when a new heartbeat arrives, the inter-arrival time is put into a sampling sliding window, and at the same time, the previous oldest one is pushed out of the sampling window. Then the arrival time in the sampling window is used to compute the distribution of inter-arrival times, and get the average inter-arrival time Δt in this sliding window. After that, based on Equations (11-13), we compute the current value of timeout τ, which gives the next freshness point (see Fig. 2). Applications will perform some actions, or start to suspect the process by comparing the τ value and its current heartbeat arrival time (see Fig. 2).

We are unable to get the communication delay from the sender to the receiver when it is lost (see the second case in Fig. 2). In order to ensure the effectiveness of the proposed approach, and considering the influence of message loss, we use the time series theory to fill in the gap. In detail, we fill in the gaps with a value computed by $d_i = (\Delta t \cdot \overline{nag}) + d_{i-1}$, where $\overline{nag}$ is the average number of observed adjacent gaps [18].

The detailed information for the implementation of SFD is shown in Algorithm 2. Here we first set some initial parameters, including an initial safety margin value for SM_1. After that, SFD could get the feedback information (Step 2 in Algorithm 2): If SM_1 is the proper parameter for SFD to obtain the expected output QoS, then the feedback information is 0, and the SFD is stable. This means the current parameters are proper for the network system; If SM_1 is not the exact proper parameter for SFD to acquire the expected output QoS and the output QoS matches the control rules, then the feedback information is $\pm\beta$ based on specific output QoS status; If SM_1 is not the exact proper parameter for SFD to acquire the expected output QoS and the output QoS does not match the control rules, then the SFD give a response about this mistake (all the possible values are not proper for SM_1). Finally, if the SFD does not show "give a response", SFD adjusts the parameter SM until gets the expected output QoS.

For Chen FD [28], they have to find the exact proper parameter value for its initial safety margin to achieve the expected output QoS (because they could not automatically adjust the parameter); Otherwise, the output QoS can not satisfy the $\overline{QoS}$ (users' requirements). The ϕ FD [30-31] and Bertier FD [29, 33] also have same drawback, which is solved by our SFD.

Algorithm 2 A Method to Adjust the Parameters

```
1: Begin
2: Initialization:
3:     $\overline{T_D}$: Set the detection time;
4:     $\overline{MR}$: Set the mistake rate;
5:     $\overline{QAP}$: Set the query accuracy probability;
6:     Set the initial safety margin value for $SM_1$;
7:     Set the constant parameters $\alpha$, and $\beta$;
8: Step 1: Get the relevant data
9:     Get the output QoS ($T_D$, MR, QAP).
10: Step 2: Get the feedback information
11:     If $T_D > \overline{T_D}$, $MR < \overline{MR}$, and $QAP > \overline{QAP}$: $Sat_k\{QoS, \overline{QoS}\} = \beta$;
12:     If $T_D < \overline{T_D}$, $MR < \overline{MR}$, and $QAP > \overline{QAP}$: $Sat_k\{QoS, \overline{QoS}\} = 0$;
13:     If $T_D < \overline{T_D}$, $MR > \overline{MR}$, and $QAP < \overline{QAP}$: $Sat_k\{QoS, \overline{QoS}\} = -\beta$;
14:     Others (for example, if $T_D > \overline{T_D}$, and $MR > \overline{MR}$): "Give a response" (this SFD can not get this high QoS requirement), and stop SFD (go to line 18).
15: Step 3: Adjust parameters
16:     Send $Sat_k\{QoS, \overline{QoS}\}$ to the SFD;
17:     Adjust SFD relevant parameters based on the value of $Sat_k\{QoS, \overline{QoS}\}$;
18: End
```

V. Performance Evaluation

In order to demonstrate that the presented general non-manual analysis method in this paper is effective, we evaluate and comparatively analyze the performance of SFD, ϕ FD [30-31], Chen FD [28], and Bertier FD [29, 33] in general experimental environments (seven WAN cases: all are real data, one is obtained between Japan and Switzerland, the others are from PlanetLab), which are representative as general cloud computing networks (see Fig. 1).

The experiments are performed based on the model in Fig. 2. One (process p) sends heartbeat messages periodically to the other one (process q) for an arbitrarily long period, and the other one q receives the messages from process p. In each experiment, the heartbeat sending and arrival times are logged into the log files in the monitoring computer q. These logged arrival time is used to replay the execution for each FD scheme. That implies all the FDs are compared in the same experimental condition: the same network model, the same heartbeat traffic, and the same experiment parameters (sending interval, sliding window size, communication delay, input, etc.). Thus, it provides a fair experimental platform for every FD. The logged sending time is used only for statistics. All heartbeat messages use the UDP/IP protocol. A low-frequency *ping* process runs in parallel with the experiment as a means to obtain a rough estimation of the round-trip time, and also to make sure the network is connected.

In the experiments, each FD scheme uses a sliding window to save past samples for computing future estimations. All the experiments for the four FDs use the same fixed window size ($WS = 1,000$). It is reasonable to analyze the sampled data only after the sliding window is full because the network is unstable during the warm-up period. In these experiments, the heartbeats are generated at a target rate of one heartbeat every 100 ms.

The main parameters are as follows: In order to find the best QoS and compare with the others, here we set $SM_1 = \alpha$ for SFD; For Chen FD, the parameters are set the same as in [28]: $\alpha \in [0, 10000]$; For ϕ FD, the parameters are set the same as in [30-31]: $\Phi \in [0.5, 16]$; For Bertier FD, the parameters are set the same as in [29, 33]: $\beta = 1$, $\phi = 4$, $\gamma = 0.1$. In each experiment, the other basic experimental parameters of FDs are the same.

In these experiments, after discarding some initial period, we have measured the following three key QoS metrics for the entire execution: T_D, MR, QAP. It is not easy to compare parametric failure detectors because, depending on the value set for their parameter, their behaviors can be completely different. A common mistake is to set some arbitrary values for the parameters and then compare two parametric failure detectors based on the measured detection time and accuracy. This almost always leads to the erroneous conclusion that one is better for detection time while the other provides higher accuracy.

In contrast, we use the developed approach when conducting experiments for the ϕ FD [30-31]. The idea is based on the following question: given a set of QoS requirements, can the failure detector be parameterized to match these requirements? To answer this question, we consider a space of QoS defined by the detection time on one axis and an accuracy metric (e.g., MR, or QAP) on the other axis. Then, we measure the area covered by the failure detector when we vary its parameter from a highly aggressive behavior to a very conservative one (i.e., T_D becomes larger and larger[9].). The area covered by a failure detector is the area that corresponds to a set of QoS requirements that can possibly be matched by that failure detector. In each experiment, different output values (MR, QAP and T_D) were obtained from the following respective parameters: For SFD, a list about the initial safety margin SM_1 is given, and other parameters including α and β are not listed for that they only impact the rate of self-tuning adjustability; For Chen FD, a list about the initial safety margin is given; For ϕ FD, a list about the threshold parameter Φ is given.

We find that when the parameter continuously changes in sequential order (for example, from the small value to the large value), the graph is serially developing, and we can obtain plenty of points, which can be fitted on this curve graph.

A. Experiment in a WAN

This experiment involves two computers: one was located at the Swiss Federal Institute of Technology in Lausanne (EPFL), in Switzerland, and the other was located in JAIST, Japan. The two computers communicate with each other through a normal intercontinental Internet connection.

[9]Here each point in the graph is corresponding to a parameter in this FD scheme. When we choose the parameter in order (for example, from the small value to the large value), we could get a lot of points which can be fitted with a serial curve.

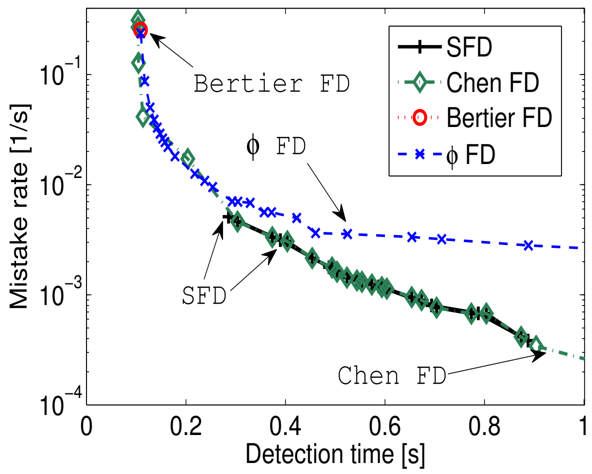

Fig. 6. Mistake rate vs. detection time in a WAN.

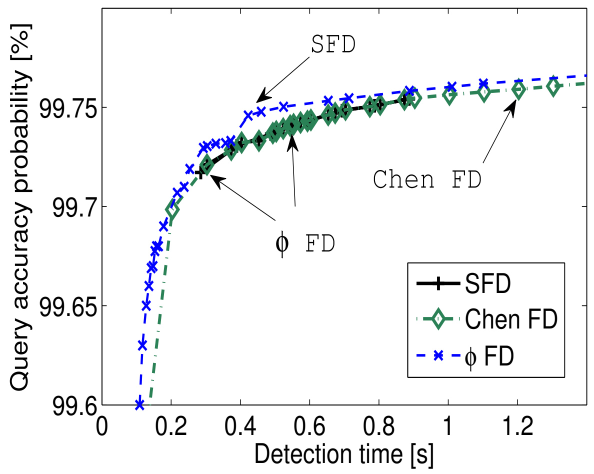

Fig. 7. Query accuracy probability vs. detection time in a WAN.

In this experiment, we use exactly the same trace files from the paper about ϕ FD [30-31], and these trace files are publicly available on the lab website [40]. Therefore, this provides a common ground for evaluating the performances of SFD, Chen FD [28], Bertier FD [29, 33], and ϕ FD [30-31].

1) Experiment Setting: Hardware/software/network: In detail, the trace files and relevant data were obtained from the following experiment setting.

Heartbeat sampling The experiment was over in one week (started on April 3 at 2:56 UTC, and finished on April 10 at 3:01 UTC). During the experimental period, the average sending rate actually measured was one heartbeat every $103.501\ ms$ (standard deviation: 0.189 ms; min.: 101.674 ms; max.: 234.341 ms). Furthermore, only $5,822,521$ out of the $5,845,713$ heartbeat messages sent out were received, thus the message loss rate was about $0.399\,\%$. After checking the trace files more carefully, the messages losses were due to 814 different bursts. The majority of total bursts were short length bursts, while the maximum burst-length was $1,093$ heartbeats (only one), and it lasted about 2 minutes. Furthermore, most of the heartbeats were not directly from Asia to Europe, but actually, routed through the United States.

Round-trip time The average RTT is 283.338 ms, with a standard deviation of 27.342 ms, a minimum of 270.201 ms, and a maximum of 717.832 ms. By analyzing the trace files, we found the average CPU load rate for the sending host, and the receiving hosts were $1/67$ and $1/22$, respectively, so they were below the full capacity of the computers.

2) Experimental Results: The experimental results for T_D, MR and QAP are shown in Figs. 6-7. Fig. 6 shows MR comparison of FDs, where the vertical axis is on a logarithmic scale. The best values are located at the lower left corner, which means this FD provides short detection time and has a low mistake rate. Fig. 7 shows QAP comparison of FDs, where the vertical axis is on a linear scale. The best values are located at the upper left corner, which means that the FD provides short detection time with a high QAP.

In Fig. 6, when $T_D < 0.3$ s, the Chen FD and ϕ FD can obtain the similar MR and T_D. When 0.3 s$< T_D < 0.9$ s, SFD and Chen FD have similar results and are slightly better than ϕ FD. When $T_D > 0.9$ s, Chen FD can obtain the lowest MR with the same T_D. For SFD, there is no data in the too aggressive range ($T_D < 0.3$ s) and the too conservative range ($T_D > 0.9$ s). Since SFD can dynamically adjust its parameters.

At the beginning, the initial safety margin SM_1 was set as a very small value, then output QoS in our SFD had a short detection time T_D ($T_D < \overline{T_D}$) and a large mistake rate MR ($MR > \overline{MR}$). It implies the output QoS does not satisfy the $\overline{QoS}$, and we should take multiple steps to increase SM in order to reduce the MR. Then, our scheme gradually increased SM in next multiple freshness points τ to reduce the MR of output QoS. Eventually, we satisfy the $\overline{QoS}$ for the application.

For the next SM_1 value, which is slightly larger than the former one[10], SFD first had a lower MR than the former one (with a lower SM_1 value), which was still larger than $\overline{MR}$. Then our scheme multiple increases SM to gradually get lower MR of output QoS, as far as the output QoS satisfies the $\overline{QoS}$. So for the total output QoS, usually a larger SM_1 leads to a larger T_D, a lower MR and a higher QAP. But it is not absolute, because the SFD could automatically adjust the SM to satisfy the $\overline{QoS}$.

Interestingly, there are some bursts in this WAN experiment, so for every SM_1 value, after the output QoS of SFD has been adjusted to satisfy the $\overline{QoS}$, due to the bursts, there were some fluctuations for the output QoS of SFD.

For those SM_1 values with $T_D > 0.9$ s in SFD, our scheme can reduce the SM in next freshness point τ to get shorter T_D gradually, though it leads to slightly larger MR. If the first output T_D is greater than 0.9 s in SFD (because the SM_1 is very large), we should spend more steps to reduce the SM to reduce T_D, and let $T_D < \overline{T_D}$ eventually. Then in total performance, output MR is larger and T_D is shorter. It is reasonable for those SM_1 values with $T_D > 0.9$ s that, when SFD's SM_1 increases, T_D becomes larger and MR becomes lower. The graphs of ϕ FD are stopped early (at 2.43 s), due to the rounding error preventing the graphs to the very conservative case. For Bertier FD, it has only one point because it has no dynamic parameters. This is the same status in other experimental environments. In Fig. 7, we have similar

[10]In our SFD experiments, SM_1 gradually increases from a list of possible values.

TABLE I
SUMMARY OF THE WAN EXPERIMENTS

WAN case	Sender	Sender-hostname	Receiver	Receiver-hostname
WAN-1	USA	planet1.scs.stanford.edu	Japan	planetlab-03.naist.ac.jp
WAN-2	Germany	planetlab-2.fokus.fraunhofer.de	USA	planet1.scs.stanford.edu
WAN-3	Japan	planetlab-03.naist.ac.jp	Germany	planetlab-2.fokus.fraunhofer.de
WAN-4	China	planetlab2.ie.cuhk.edu.hk	USA	planet1.scs.stanford.edu
WAN-5	China	planetlab2.ie.cuhk.edu.hk	Germany	planetlab-2.fokus.fraunhofer.de
WAN-6	China	plab1.cs.ust.hk	Japan	planetlab1.sfc.wide.ad.jp

TABLE II
SUMMARY OF THE EXPERIMENTS: STATISTICS

Experiment	Heartbeats		Heartbeats period				RTT
case	total (#msg)	loss rate	send (Avg.)	send (stddev)	receive (Avg.)	receive (stddev)	(Avg.)
WAN-1	$6,737,054$	0%	12.825 ms	13.069 ms	12.83 ms	14.892 ms	193.909 ms
WAN-2	$7,477,304$	5%	12.176 ms	1.219 ms	12.206 ms	19.547 ms	194.959 ms
WAN-3	$7,104,446$	2%	12.21 ms	1.243 ms	12.235 ms	4.768 ms	189.44 ms
WAN-4	$7,028,178$	0%	12.337 ms	9.953 ms	12.346 ms	22.918 ms	172.863 ms
WAN-5	$7,008,170$	4%	12.367 ms	15.599 ms	12.94 ms	16.557 ms	362.423 ms
WAN-6	$7,040,560$	0%	12.33 ms	10.185 ms	12.42 ms	17.56 ms	78.52 ms

results to the Fig. 6.

B. Extensive PlanetLab WAN Experiments

We have analyzed the behavior of the implementation of the SFD in a large collection of environments. Here we focus on the most relevant WAN environments in order to obtain the general experimental analysis. The main goal of our experiments was to observe the performance of the SFD to be automatically configured well to match the $\overline{QoS}$. In this case, we compare SFD against Chen-FD, Bertier-FD, and ϕ-FD. Here we first describe the various WAN environments, and then make comparison between SFD and existing FDs using feasible methods, and finally the relevant results are discussed.

1) Experiment Settings: The experimental environments are described below, and the corresponding statistics are summarized in Table I and Table II. Here six representative WAN experiments were conducted on the PlanetLab (http://www.planet-lab.org/), using nodes located in USA, Europe (Germany), Japan, and China (Hong Kong). Each location communicates with the other three locations (see Fig. 8 and Tables I-II). The locations and hostnames are summarized in Table I. Each WAN experiment was set to last for about 24 hours, with a target heartbeat interval set to 10 ms.

Environment 1 (WAN-1). This set was from Stanford University, USA to Nara Institute of Science and Technology, Japan, starting from March 12, 2007. The effective heartbeat interval was 12.825 ms, and total 6, 737, 054 heartbeats were sent. The mean heartbeat arrival time was 12.83 ms (thus showing a slight clock drift) with a standard deviation of 14.892 ms. The average round-trip time was 193.909 ms.

Environment 2 (WAN-2). This set was from Germany to USA, starting from March 8, 2007. Total 7, 477, 304 heartbeats were sent, and the loss rate was 5%.

Environment 3 (WAN-3). This set was from Japan to Germany, starting from March 6, 2007. Total 7, 104, 446 heartbeats were sent, and the loss rate was 2%.

Environment 4 (WAN-4). This set was from Hong Kong (China) to USA, starting from March 10, 2007. Total 7, 028, 178 heartbeats were sent, and the loss rate was 0%.

Environment 5 (WAN-5). This set was from Hong Kong (China) to Germany, starting from March 11, 2007. Total 7, 008, 170 heartbeats were sent, and the loss rate was 4%.

Environment 6 (WAN-6). This set was from Hong Kong University of Science and Technology, Hong Kong to Mural Labolatory, Keio University Shonan Fujisawa Campus, Japan. Total 7, 040, 560 heartbeats were sent, and the loss rate was 0%.

As a final note, we use the ping process conducted in parallel with the experiments to obtain the ping log files, which demonstrate that the Network was always connected during experiments.

2) Experimental Results and Discussions: A similar behavior can be observed in the different experimental settings. The experimental results from WAN-2 to WAN-6 obtained on the PlanetLab are similar to WAN-1. For the limited space for this paper, here we only show the experimental results from WAN-1 in Figs. 9-10.

We observe the results obtained in the experiment WAN-1 between USA and Japan (see Figs 9-10). Bertier FD behaves as an aggressive failure detector with only one point because it has no dynamic parameters. While Chen FD is a conservative failure detector, and can get the 0 MR finally. For ϕ FD, the rounding errors prevent to compute points in the conservative range, and the curve for ϕ FD stops with a T_D of 1.58 s, a MR of 0.002, and a QAP of 99.8%. While the three schemes have similar beginning point, none of these schemes could automatically adjust parameters based on the dynamic network case. Obviously, it is solved by SFD.

The curve for SFD is from 0.10 s to about 0.87 s. First it obtains the beginning point with a T_D of 0.10 s, a MR of 0.31, and a QAP of 99.5%. The SFD curve gradually changes to the point (a T_D of 0.87 s, a MR of 0.00041, and a QAP

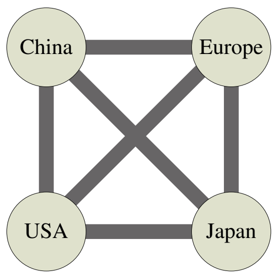

Fig. 8. PlanetLab WAN Hosts

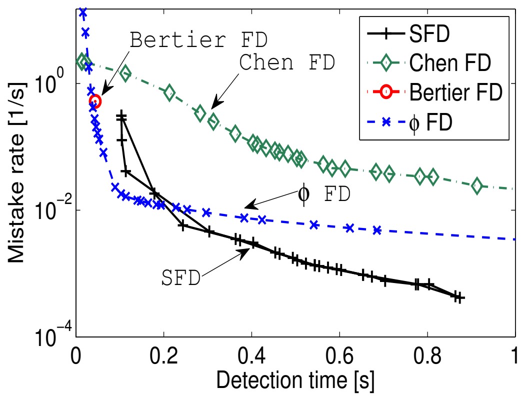

Fig. 9. WAN1: MR vs T_D

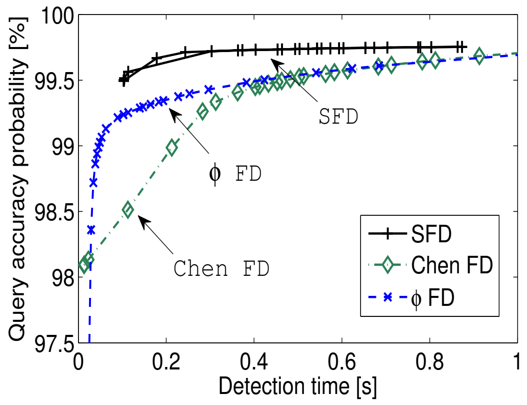

Fig. 10. WAN1: QAP vs T_D

of 99.8%) with the parameter SM_1 rising. After that, when initial safety margin rises again, Chen FD will get longer T_D in its output QoS, while SFD does not. Because SFD finds this output T_D is larger than the requirement, it automatically adjusts the other parameters (by setting $Sat\{QoS, \overline{QoS}\} = -\beta$ to reduce SM) to reduce the T_D, though leads to the expenditure of larger MR. During this adjusting period, SFD should repeatedly adjust the parameter $Sat\{QoS, \overline{QoS}\} = -\beta$ to achieve smaller output T_D gradually. Finally, the output T_D has $T_D < \overline{T_D}$ with the expected output MR and QAP ($MR < \overline{MR}$ and $QAP > \overline{QAP}$). With increasing value of SM_1, the SFD graph gradually develops to the point with a T_D of 0.10 s, a MR of 0.0.31, and a QAP of 99.5%, which is very close to the beginning point.

The optimization of SFD over ϕ FD, Bertier FD, and Chen FD is significant: Although sometimes SFD does not provide the better performance than other FDs (for example, when D_T is smaller than about 0.2 s, ϕ FD get lower MR than SFD does at the same D_T), yet only SFD could automatically adjust its parameters to match the $\overline{QoS}$. Specifically, with initial safety margin SM_1 rising, *SFD can always automatically adjust $Sat\{QoS, \overline{QoS}\}$ and find suitable SM to get a satisfactory output QoS for users' requirements. In contrast, other schemes could blindly provide different output QoS*: some are just temporarily suitable for the requirements; yet some are never, then engineers have to manually change the relevant parameters. *These schemes must try all the possible parameter values, and get a performance output graph to know which parameter values are acceptable for the network (manually choose relevant parameters). If the network has significant changes, the engineers have to change the relevant parameters manually again.* In conclusion, SFD has good self-tuning property, and it can be more effectively used in cloud computing environments for extensive industrial and commercial usage.

The parameter settings in each FD are key factors, since a different parameter setting could lead to a completely different results. For a qualitative analysis, when the parameter continuously changes in a fixed order (for example, from the small value to the large value), the graph serially monotonously develops in most cases[11]. There is no quantitative relationship between the parameter change and the results change, because they are independent.

[11]The possible reason for the abnormal cases is that some burst data and too old data affect the output QoS.

In summary, we evaluate the performance of SFD in a variety of WAN environments (see Fig. 8). Experimental results have shown that the SFD can automatically adjust parameters to provide good performance at general network cases over other state-of-the-art failure detectors.

C. Comparative Analysis of the Four FDs

All our above experiments cover the most representative application environment found at present. Based on them, we conclude the following:

Self-tuning property: The experiments demonstrate that SFD outperforms the existing FDs (ϕ FD [30-31], Bertier FD [29, 33], and Chen FD [28]) in terms of self-tuning capacity. Specifically, in cloud computing networks, with initial safety margin SM_1 rising, SFD can always automatically adjust $Sat\{QoS, \overline{QoS}\}$ and find suitable SM to get a satisfactory output QoS for users' requirements. In contrast, other schemes could blindly provide different output QoS. If the network has significant changes, the engineers have to change the relevant parameters manually again.

Effect of window size: We analyze the effect of window size on QoS of FDs. For ϕ FD, a larger window size tends to achieve better performance [41]. The possible reason is that historical information is important for ϕ FD to get good QoS. ϕ FD is based on the normal distribution function, so the larger size window could obtain more historical data, and so compute more adaptive normal distribution function for the relevant network case. For Bertier FD, the effect of window size on their QoS can be negligible. The possible reason is that Bertier FD has no tuning parameters. For Chen FD and SFD, a lower window size leads to better performance [41]. Chen FD is based on Functions (2) and (3), and SFD is based on Functions (2) and (11-13). When window size increases, there are more historical data for Chen FD and SFD based on Function (2), and the many burst data and too old data may affect the output QoS, making less ideal contributions (even bad contributions) for achieving good performance. Chen FD and SFD take less time to adapt the dynamic network with a reduced window size.

In summary, SFD is a practical self-tuning FD which can be effectively used for industrial and commercial application to automatically satisfy QoS for users' requirements. Furthermore, SFD has good scalability. Because it is able to get acceptable performance with very small window size, and it can save valuable memory resources. All the evidence supports our conclusion that the general self-tuning failure detection analysis of SFD is effective.

VI. Conclusion and Future Work

Services in the distributed networks may be virtualized with specific servers. Some of the servers are active and available, while others are busy or heavy loaded, and the remaining are offline for various reasons. Users would expect the right and available servers to complete their application requirements. Therefore, in order to provide an effective control scheme and parameters guidance for service conditions, failure detection is essential to meet users' service expectations. Most existing Failure Detector (FD) schemes do not automatically adjust their detection service parameters for the dynamic network conditions, thus they couldn't be used for actual application. This paper explores FD properties with relation to the actual and automatic fault-tolerant cloud computing networks, and find a general non-manual analysis method to self-tune the corresponding parameters to satisfy users' requirements. Based on this general automatic method, we propose a specific and dynamic Self-tuning Failure Detector scheme, called SFD, as a major breakthrough in the existing schemes. We carry out actual and extensive experiments to compare the quality of service performance between the SFD and several other existing FDs. Experimental results demonstrate that our scheme can automatically adjust SFD control parameters to obtain corresponding services and satisfy user requirements, while maintaining good performance. Such an SFD can be extensively applied to industrial and commercial usage, and it can also significantly benefit the cloud computing networks. Our SFD is also appropriate for the "one monitors multiple" and "multiple monitor multiple" cases based on the parallel theory. In our future work, we would like to explore software engineering solutions in the United States southern states education cloud consortium (see Fig. 1), specifically addressing mechanisms for ensuring the QoS for resource access and service allocations in multi-cloud scenarios.

References

[1] J. Lischka, H. Karl. A Virtual Network Mapping Algorithm based on Subgraph Isomorphism Detection. In *Proc. SIGCOMM 2009 workshop* on VISA: Virtualized Infrastructure Systems and Architectures, pp. 81-88, Barcelona, Spain, August 17-21, 2009.

[2] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration. *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17-29, 2008.

[3] L. M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner. A break in the clouds: towards a cloud definition. SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50-55, Dec. 2008.

[4] Virtual Computing Lab production site: http://vcl.ncsu.edu

[5] Virtual Computing Lab Apache software foundation: http://incubator.apache.org/projects/vcl.html

[6] N. Xiong, A. Vandenberg, M. L. Russell, and K. P. Robinson. A Multi-Cloud Computing Scheme for Sharing Computing Resources to Satisfy Local Cloud User Requirements Networks. *Proc. Third International Conference on the Virtual Computing Initiative (ICVCI3)*, no. 1, pp. 1-10, North Carolina, Oct. 22-23, 2009.

[7] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, S. Luz. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *Proc. SIGCOMM 2008*, pp. 75-86, Seattle, USA, August 17-22, 2008.

[8] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui. A realistic look at failure detectors. In *Proc. Intl. Conf. on Dependable Systems and Networks (DSN'02)*, pp. 345–353, Washington DC, Jun. 2002.

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

[10] I. Gupta, T. D. Chandra, G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proc. 20th ACM symp. on Principles of Distributed Computing*, pp. 170–179, Newport, Rhode Island, United States, Aug. 26–29, 2001.

[11] F. Silveira, C. Diot, N. Taft, R. Govindanz. ASTUTE: Detecting a Different Class of Traffic Anomalies. *SIGCOMM'10*, New Delhi, India, August 30–September 3, 2010.

[12] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage. California Fault Lines: Understanding the Causes and Impact of Network Failures. *SIGCOMM'10*, New Delhi, India, August 30–September 3, 2010.

[13] W. Terpstra, J. Kangasharju, C. Leng, A. . Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. *SIGCOMM'07*, Kyoto, Japan, August 27-31, 2007.

[14] R. S. Bhatia, M. Kodialam, T. V. Lakshman, S. Sengupta. Bandwidth Guaranteed Routing With Fast Restoration Against Link and Node Failures. *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1321-1330, Dec. 2008.

[15] S. Stefanakos. Reliable Routings in Networks With Generalized Link Failure Events. *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1331-1339, Dec. 2008.

[16] M. Bozinovski, H. P. Schwefel, and R. Prasad. Maximum Availability Server Selection Policy for Efficient and Reliable Session Control Systems. *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 387-399, Apr. 2007.

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

[18] R. C. Nunes, I. Jansch-Porto. Modeling communication delays in distributed systems using time series. In *Proc. 21st IEEE Symp. on Reliable Distributed Systems (SRDS'02)*, pp. 268-273, Suita, Japan, Oct. 2002.

[19] K. Takeuchi, T. Tanaka, and T. Yano. Asymptotic Analysis of General Multiuser Detectors in MIMO DS-CDMA Channels. *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 3, pp. 486-496, 2008.

[20] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subram. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proc. SIGCOMM 2009*, pp. 39-50, Barcelona, Spain, August 17-21, 2009.

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

[22] M. K. Aguilera, W. Chen, and S. Toueg. Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science*, 220(1): 3–30, Jun. 1999.

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

[24] R. Guerraoui, M. Larrea, and A. Schiper. Non-blocking atomic commitment with an unreliable failure detector. *Symposium on Reliable Distributed Systems*, pp. 41–50, 1995.

[25] M. Larrea, A. Fernandez, and S. Arevalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC-00)*, pp. 334–334, NY, Jul. 2000.

[26] M. T. Hajiaghayi, N. Immorlica, V. S. Mirrokni. Power Optimization in Fault-Tolerant Topology Control Algorithms for Wireless Multi-hop Networks. *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1345-1358, Dec. 2007.

[27] D. Leonard, Y. Zhongmei, V. Rai, D. Loguinov. On Lifetime-Based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks. *IEEE/ACM Transactions on Networking*, vol. 15, no. 3, pp. 644-656, June 2007.

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

[29] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'02)*, pp. 354–363, Washington, DC, USA, June 2002.

[30] N. Hayashibara, X. Défago, R. Yared, and T. Katayama. The ϕ accrual failure detector. In *Proc. 23rd IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'04)*, pp. 66–78, Florianpolis, Brazil, Oct. 2004.

[31] X. Défago, P. Urban, N. Hayashibara, T. Katayama. Definition and specification of accrual failure detectors. In *Proc. Intl. Conf. on Dependable Systems and Networks (DSN'05)*, pp. 206–215, Japan, Jun. 2005.

[32] A. Basu, B. Charron-Bost, and S. Toueg. Simulating Reliable Links with Unreliable Links in the Presence of Process Crashes. In *Proc. Workshop on Distributed Algorithms (WDAG 1996)*, pp. 105–122, Italy, 1996.

[33] M. Bertier, O. Martin, P. Sens. Performance analysis of a hierarchical failure detector. In *Proc. Dependable Systems and Networks (DSN'03)*, pp. 635–644, San Fra., USA, Jun. 2003.

[34] R. Macêdo. Implementing failure detection through the use of a self-tuned time connectivity indicator. TR, RT008/98, Laboratŕio de Sistemas Distribudos - LaSiD, Salvador-Brazil, Aug. 1998.

[35] P. Felber. *The CORBA object group service - a service approach to object groups in CORBA*. PhD thesis, Départment Dínformatique, Lausanne, EPFL, Swizerland, 1998.

[36] 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. 21st IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'02)*, pp. 244–249, Osaka, Japan, Oct. 2002.

[37] P. Urbán, I. Shnayderman, and A. Schiper. Comparison of failure detectors and group membership: Performance study of two atomic broadcast algorithms. In *Proc. IEEE Intl. Conf. on Dependable Systems and Networks (DSN'03)*, pp. 645–654, CA, USA, June 2003.

[38] M. Müller. Performance evaluation of a failure detector using SNMP. Semester project report, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, Feb. 2004.

[39] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM '88*, pp. 314–329, Stanford, CA, USA, Aug. 1988.

[40] *http://ddsg.jaist.ac.jp/en/jst/data.html.*

[41] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, Y. Ganjali, C. Diot. Characterization of failures in an operational IP backbone network. *IEEE/ACM Transactions on Networking*, vol. 16, no. 4, pp. 749-762, Aug. 2008.

APPENDIX

Proof of Theorem 1 A failure detector of class $\Diamond P$, must verify the two properties represented by Theorem 2 and Theorem 3.

Theorem 2 *Strong Completeness. Eventually, every process that crashes is permanently suspected by every correct process.*

$$\exists t_0, \forall t \geq t_0, \forall p \in correct(t), \forall q \in crashed, q \in suspected_p(t).$$

Theorem 3 *Eventually strong accuracy. There is a time after which a correct process is no longer suspected by any correct process.*

$$\exists t_{bound}, \forall t \geq t_{bound}, \forall p, q \in correct(t), q \notin suspect_p(t).$$

Strong Completeness

There is a time t_{mute} after which no correct process q receives heartbeat messages from the crashed process p, and there is a time $t_{timeoutk}$ after which all correct processes q permanently suspect p.

Lemma 1. *If process p crashed at t_{crash}, then there is a time t_{mute} after which process q stops receiving messages from p.*

$$t_{mute} \leq t_{crash} + \Delta_{msg},$$

where Δ_{msg} means the maximum time after GST, from the sending of a message, to the delivery, and to processing by its destination process, we assume that the destination process has not failed.

Proof: All the time instants considered in the rest of this section are assumed to be after GST. We also assume that, at these instants, all the messages sent before GST have already been delivered and processed. These assumptions allow us to consider in the rest of the section, that the unknown bounds on process speeds and on message transmission times, hold.

$$\exists t_{GST} : \forall m_k | t_{sk} \geq t_{GST} : (t_{rk}) - t_{sk} < \Delta_{msg},$$

where t_{sk} is the time when p sends m_k, and t_{rk} is the time when q receives m_k.

Suppose a process p crashed at t_{crash}. Then, p stops sending heartbeat messages.

$$\nexists m_k \mid t_{sk} \geq t_{crash}$$

The process q cannot receive message k from process p after $t_{rk} + \Delta_{msg}$. Hence, process q cannot receive any message from process p after $t_{rk} + \Delta_{msg}$.

Lemma 2 *For any sequence of k messages received by process q from p, there is a time τ_k after which process q starts suspecting process p if it does not receive any message from p.*

From *Step 2 in Algorithm 2*, when the process q receives a message m_{k-1} from process p, process q calculates a new τ_k after which process q starts suspecting process p. We must prove that the τ_k is always bounded. The τ_k is calculated as follows (Algorithm 2):

$$\tau_k = SM_{(k)} + EA_{(k)}, \tag{14}$$

For this algorithm, if some process (p) crashed at time t_{crash}, then the other processes (q) cannot receive any heartbeat from it. And p will be added into the *suspect list* (Step 2 in Algorithm 2) and will never be removed from the list after t_{crash}. Therefore, p is permanently suspected by every correct process.

In Step 2, if $k > WS$, WS is the window size. EA_k is equivalent at:

$$EA_k = \frac{1}{WS}(\sum_{i=k-WS-1}^{k} t_{ri} - \Delta_i * i) + k\Delta_i.$$

From our model

$$EA_k < \frac{1}{WS}(\sum_{i=k-WS-1}^{k} t_{si} + \Delta_{msg} - \Delta_i * i) + k\Delta_i$$

as $t_{si} = t_{s(i-1)} + \Delta_{msg} + k\Delta_i$, then

$$EA_k < t_{s0} + \Delta_{msg} + k\Delta_i. \tag{15}$$

The expected arrival time of the m_k message is bounded by:

$$t_{sk} < EA_k \leq t_{sk} + \Delta_{msg} \tag{16}$$

From Functions (12-13), the safety margin SM is obtained:

$$SM_k = SM_{(k-1)} + Sat_k\{QoS, \overline{QoS}\} \cdot \alpha,$$

and

$$Sat_k\{QoS, \overline{QoS}\} = \begin{cases} \pm\beta, QoS > \overline{QoS}; \\ 0, QoS \leq \overline{QoS}. \end{cases}$$

Here, we assume $\alpha < min\{SM_1, SM_2, SM_3, ..., SM_k\}$,

$$\because 0 < \beta < 1, \therefore SM_2 = SM_1 + Sat_1\{QoS, \overline{QoS}\} \cdot \alpha,$$

and we also have $SM_1 - \beta\alpha \leq SM_2 \leq SM_1 + \beta\alpha$, $\because SM_1 - \beta\alpha > 0$, then $0 < SM_2 \leq SM_1 + \beta\alpha$. Based on the above, we have

$$0 < SM_k \leq SM_{(k-1)} + \beta\alpha. \tag{17}$$

From Functions (16-17), one has $0 < SM_3 \leq SM_1 + 2\beta\alpha$, $0 < SM_4 \leq SM_1 + 3\beta\alpha$, ...

$$0 < SM_k \leq SM_1 + (k-1)\beta\alpha. \tag{18}$$

From Function (16), we show that the expected arrival date for any message m_k is bounded; and from Function (18), the safety margin SM_k is bounded. All components of τ_k in Function (16) are bounded, so we can deduce that τ_k is bounded. If for each message m_{k-1} received from process p, process q activates a bound timeout, then there is a time after which q suspects p, if it receives no new message from p. Then, strong completeness is proved.

Eventually strong accuracy

Theorem 3 is verified if the τ_k of process q is large enough to avoid the situation where process q wrongly suspects process p.

Lemma 1. *Every time q times out and p is correct, β is increased. There is a time t_{bound} where safety margin SM is large enough to avoid false detection, and SM stops increasing. When SM becomes higher than Δ_{msg}, then no false detection can occur.*

$$\exists t_{bound}, \forall t \geq t_{bound}, SM(t) \geq \Delta_{msg}, and\ \ SM(t) = SM(t+1).$$

Lemma 2. *There is a time after which τ_k is greater than ($t_{sk} + \Delta_{msg}$).* $\exists t_{bound}, \forall t \geq t_{bound}, \tau_k \geq t_{sk} + \Delta_{msg}$

Proof: From Lemma 1, 2 and 3, we can say that $\forall m_k, t_{sk} < EA_k$. With β increased, there is a time t_{bound}, after which $SM(t) \geq \Delta_{msg}$. Therefore, we can conclude that $\exists t_{bound}, \forall t > t_{bound}, \tau_k > t_{sk} + \Delta_{msg}$. Theorem 3 is verified because if τ_k is larger than ($t_{sk} + \Delta_{msg}$), then process p cannot be considered by process q as having failed. □