A Failure Detection Approach of Data Center Networks*

Kai Shen†
School of Software
Shanghai Jiao Tong University
Wallamaloo, New Zealand
knshen@sjtu.edu.cn

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings. It is an *alternate* style which produces a *tighter-looking* paper and was designed in response to concerns expressed, by authors, over page-budgets. It complements the document *Author's (Alternate) Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2$_\epsilon$ and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file. A compiled PDF version is available on the web page to help you with the 'look and feel'.

CCS Concepts

•**Computer systems organization** → **Embedded systems;** *Redundancy;* Robotics; •**Networks** → Network reliability;

Keywords

ACM proceedings; LaTeX; text tagging

1. INTRODUCTION

Data center. The rest of this paper is organized as follows. Section 2 discusses related work,

*(Produces the permission block, and copyright information). For use with SIG-ALTERNATE.CLS. Supported by ACM.

†Dr. Trovato insisted his name be first.

WOODSTOCK '97 El Paso, Texas USA

DOI: 10.475/123_4

2. RELATED WORK

In this section, we will present various relevant works including failure detectors, failure detection architecture in distributed systems and failure detection in data center networks.

2.1 Failure Detectors

A failure detector (FD) is widely recognized as an oracle to intelligently suspect failed processes [3]. The monitored process periodically send heartbeat messages to its detector to prove its liveness.

Chen et al. proposed a FD that provides QoS [4]. It estimates the expected arrival time (EAs) of the next heartbeat message according to a slide window storing n most recent arrived messages. EAs determine the deadline that the detector will wait for the next heartbeat before suspecting the monitored process. *Chen FD* can adjust current network condition and set the timeout threshold adaptively by referring to recent heartbeats. Bertier et al. proposed a similar FD [2], whereas it uses a dynamic way to compute error margin of *Chen FD*. Other FDs like ϕ *Accrual FD* [8] and *Satzger FD* [12] can output a continuous accrual failure value other than a binary value at any time which stands for the failure probability of the process.

2.2 Failure Detection Architecture

A failure detection architecture aims to provide the service of monitoring nodes in large scale distributed systems in a scalable way. Roughly speaking, there are two kinds of architectures. The first one is hierarchical architecture [5], all the nodes are partitioned into different groups, each group has a leader node. Within a group, the leader node is responsible for monitoring all the nodes. Leader nodes periodically send node status information of his group to other leader nodes. Hierarchical architecture can reduce the number of heartbeat messages effectively, but it exists *Single Point of Failure* problem, when the leader node crashes, it needs to select a new leader.

Another alternative solution is Gossip-Style architecture [13]. Each node maintains a list containing the *heartbeat counter* for each node in the system. Every T_{gossip} seconds, each node firstly update *heartbeat counter* of itself and then randomly select another node to send its list to. Upon receiving a message, the node will merge two lists and update each node's *heartbeat counter* with the bigger one. If the *heartbeat counter* does not update after T_{fail} seconds, the node will be marked as crashed. If the *heartbeat counter* does not update after $T_{cleanup}$ ($T_{cleanup} \geq T_{fail}$) seconds,

it will be removed from the list. This approach has very low bandwidth occupation because it only produces n messages each epoch where n is the size of the system. However, with the increasing of crashing nodes, it needs more time to detect these failures.

2.3 Failure Detection in Networks

Different from the work mentioned above, failure detection in networks should consider network topology and localize failure positions at IP layer. Gill et al. present the first large-scale survey on failures in data center networks [6].

Also, there exists a large body of work on applying statistical approaches to failure localization in networks. Sherlock [1] localizes failures to limited network components by capturing dependencies between components. Shrink [10] leverages Bayesian Network model to diagnose root cause of IP network failures. Herodotou et al. [9] actively inject ping traffic into network and analyze the ping results overlaid on top of the network topology. It will generate a ranked list containing the suspected failed device and links along with their failure score that can best explain the observed ping data.

3. DATA CENTER NETWORK ARCHITECTURE

In this section, we will take a brief look at common data center network architecture.

A data center is a facility used to house computer systems and associated components, it usually contains hundreds of thousands devices and links. Modern data centers use hierarchical architecture. Figure 1 depicts a conventional data center network architecture [6, 7, 9]. At the bottom layer are racks of servers, typically there are 20 to 40 servers in each rack, and each server in the same rack connects to a Top of Rack (ToR) Switch via a 1 Gbps link. Each ToR then connects to a primary and a backup Aggregation Switch (AS) for redundancy. ASs are connected to Access Routers (ARs) forming a complete bipartite graph. Finally, ARs are connected to Core Routers (CRs).

To limit overheads and to isolate different services, servers are partitioned into virtual LANs (VLANs). All links in the data center use Ethernet as link layer protocol and physical connections are a mix of copper and fiber cables. In addition to the devices in Figure 1, there may exist other devices like load balancers (LBs) and firewalls.

Modern data center usually employs *Equal Cost MultiPath* (ECMP) routing within the data center for fault tolerance, which means that there are multiple paths from source to destination. Thus in this work, we will apply ECMP routing to the simulated network by enabling ECMP routing configurations in NS-3.

4. APPROACH OVERVIEW

In this section, we will introduce the whole design of our failure detection approach.

Different network device should have different quality of service (QoS). Gill et.al [6] have shown the diversity of different devices' failure characteristics. Unfortunately, none of the work in Section 2 treats these devices differently. We firstly divide network devices into two categories.

- **Imperative Device.** This category includes servers, ToRs and links between ToRs and servers. Typically servers run various user applications and store user data. The crash of servers may cause user-perceived faults. The crash of a ToR will make servers under it unreachable. Also Gill et.al declare that the downtime of ToRs may be very long. Thus the failure detection for these devices must be fast enough besides the accuracy requirement.

- **Non-imperative Device.** It includes links, other routers and switches. A single failure of a link or a router does not affect the functionality of a data center because of path redundancy. So administrators care more about how effective the detect service is (The ability to identify actually happening failures).

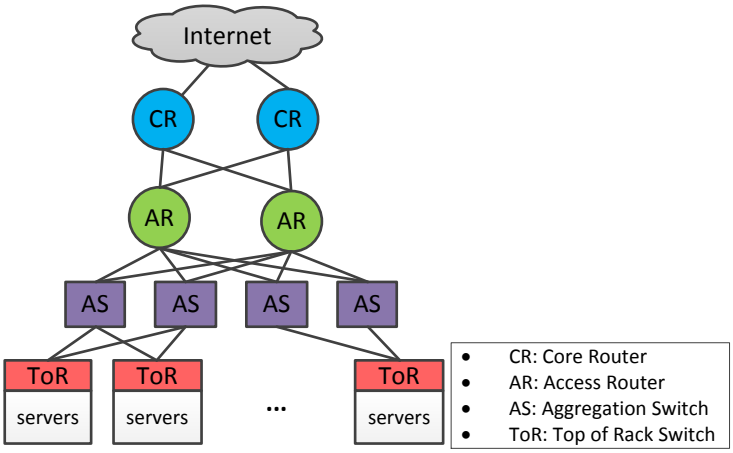

Figure 1: Conventional Data Center Network Topology

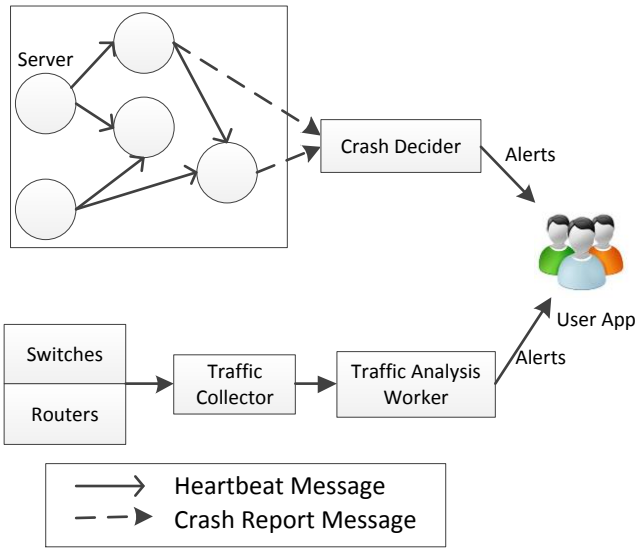

Figure 2: Failure Detection System Architecture

Figure 2 illustrates the entire architecture of our failure detection system. It mainly contains two modules. The servers failure detection system combines centralized with

decentralized structures. A server can both play the role of a detector and a monitored object. Each server is monitored by multiple FD processes installed on different servers for redundancy. A server can also monitor a couple of servers. Once the FD outputs the crash of a server, it will send a crash report message to **Crash Decider**. **Crash Decider** will determine the final status of this server.

For the detection of non-imperetative devices, **Traffic Collector** periodically collects traffic data of routers and switches and send telemetry data to **Traffic Analysis Worker**. **Traffic Analysis Worker** finally generate failure alerts by performing analysis on these data.

5. FAILURE DETECTION APPROACH

In this section, we will discuss the detailed design of different modules of the failure detection system respectively.

5.1 K-detectors Model

To detect the crash of servers and ToRs effectively and efficiently, this paper propose a detection model named K-detectors model. Our detection model is based on the assumption that once the server or ToR crashes, it never recovers by itself. The QoS of failure detection are as follows:

- **Efficiency.** It measures how fast the system can report the crash after it crashes. The system is more efficient if it takes less time to detect a failure.
- **Completeness.** It measures whether the system can detect all the crashed objects. The system is more complete if it rules out fewer true crashed ones.
- **Soundness.** In some scenarios, the behavior of the network may vary. For example, message loss and message delay may happen sometimes. The detection system should be sound enough to distinguish between such condition and a real crash.

In K-detectors model, each server is monitored by K FDs on different servers where K is a parameter tuned by users. When a FD considers a server crashes, it sends a crash report message to **Crash Decider**. If the output of the FD transforms from suspect to trust, it re-sends a message to revoke the most recent crash report.

$$S_{s,t} = \begin{cases} \text{suspect} & \text{if} \quad \|CRM_{s,t}\| \geq \lfloor (K+1)/2 \rfloor \\ \text{trust} & \text{otherwise} \end{cases} \tag{1}$$

where $CRM_{s,t}$ denotes the collection of crash report messages for server s until time t, $S_{s,t}$ denotes the status of server s at time t. If **Crash Decider** receives $\lfloor (K+1)/2 \rfloor$ crash messages of a server, it considers the server to have crashed.

5.2 Monitoring Rule Inference

To form the K-detectors model and also to meet QoS to the greatest extent, we propose the following specifications of monitoring relationships among servers:

1. Each sever monitors K servers. This balances monitor intensity of each server.
2. A server cannot monitor itself.
3. A server is monitored by K different servers.
4. $K-1$ of the K detectors are on the servers at the same rack.
5. The monitoring rule should ensure each link covered by heartbeat messages.
6. Each rack of servers monitor servers of other racks, which vary from different racks averagely.

Rationale.

Intuitively, the use of K FDs is able to adapt to bursty-traffic conditions like message loss because the determination of a failure requires at least $\lfloor (K+1)/2 \rfloor$ FDs' agreement. Also, K-detectors can prevent crashes of FDs to some extent. Theoretically, a failure can still be detected even when $K - \lfloor (K+1)/2 \rfloor = \lfloor K/2 \rfloor$ of its detectors crashed.

On the other hand, the fourth specification above can make the best of locality of LAN, which reduces the probability of degeneration on detection time in turn.

Specification five is the presupposition of link failure detection. It is easy to see that, some links cannot be distinguished from faulty links without this specification.

The last one is an extension of specification five. It is for the consideration of ToRs' detection, we will discuss it in detail in section 5.4.

5.3 FD Algorithm

Our FD algorithm is almost the same as *Chen* FD [4] except for some minor changes.

$$EA_{l+1} \approx \frac{1}{n}(\sum_{i=1}^{n} A_i - \eta s_i) + (l+1)\eta \tag{2}$$

this formula estimates the expected arriving time of next message, where s_i is the sequence number of heartbeat message, A_i is the receipt time of message i, η is message send interval. Obviously, this formula normalizes each A_i by shifting it backward ηs_i time units.

After that, we utilize hyperbolic tangent function to model current suspicious level:

$$\tau(t_{now}) = |tanh(t_{now} - EA_{l+1})| \tag{3}$$

suspicious level $\tau(t_{now})$ always ranges from 0 to 1, when t_{now} grows bigger, the suspicious level is closer to 1, which implies a higher failure probability. A continuous valued rather than binary valued output decouples the interpretation of failure data from the failure monitoring mechanism, user applications can set their own threshold according to their demands.

5.4 Failure Detection for ToRs

Apart from determination of servers' crashes, **Crash Decider** is also responsible for failure detection for ToRs. If most or even all servers in a rack is viewed as crashed, it is more likely that the top of rack switch crashes rather than the servers.

$$S_{ToR,t} = \begin{cases} \text{suspect} & \text{if} \quad \| \{s | s \in ToR\} \cap \{s | S_{s,t} = \text{suspect}\} \| \\ & \quad \geq \alpha \cdot m \\ \text{trust} & \text{otherwise} \end{cases} \tag{4}$$

where m is the number of servers per rack, α is a proportionality coefficient measuring how strict the criterion to determine failure of a ToR is. When **Crash Decider** finds more than $\alpha \cdot m$ servers under the ToR crash, it will report the crash of ToR rather than these servers. The last specification in section 5.2 ensures that once a ToR is down, each other rack has the chance to find this failure. An extreme counter-example is that if the outside detectors of rack 1 servers are all in rack 2, once ToR of rack 2 is down, **Crash Decider** never perceives the status of ToR of rack 1.

To sum up, Algorithm 1 illustrates the pseudo-code for overall detection algorithm of **Crash Decider**.

Algorithm 1 Detection Algorithm of **Crash Decider**

Input: K, η, n, α, m, τ_m
Output: $S_{i,t}$, i is an imperative device

```
1: for i in imperative devices do
2:    S_{i,t_0} ⇐ trust
3: end for
4: while True do
5:    List < Integer > failsEachRack ⇐ ∅
6:    for each i in Servers do
7:       if ||CRM_{s,t_now}|| ≥ ⌊(K + 1)/2⌋ then
8:          S_{i,t_now} ⇐ suspect
9:          rackID ⇐ getRackID(i)
10:         failsEachRack.get(rackID)++
11:      else
12:         S_{i,t_now} ⇐ trust
13:      end if
14:   end for
15:   for each i in ToRs do
16:      if failsEachRack.get(i) ≥ α · m then
17:         S_{i,t_now} ⇐ suspect
18:      else
19:         S_{i,t_now} ⇐ trust
20:      end if
21:   end for
22: end while
```

It firstly determines the status of each server (line 6 to 14), and then determines the status of ToRs (line 15 to 21).

5.5 Failure Detection for Links

The failure detection for links is another important part of our detection system, this paper presents the following QoS:

- **Precision.** Precision indicates how many suspected links are true failed.
- **Completeness.** This measures whether our approach can find all the faulty links.
- **Discrimination.** Because we will provide a short ranked list of suspected objects, this measures to what degree can our approach distinguish between normal links and faulty links.

The key idea of our approach is to capture network traffic changes over time. If the network traffic changes significantly compared with traffic in normal conditions, we believe that there exists failures somewhere in the network. The insight is that network traffic distribution is associated with routing protocol and monitoring relationships of servers, the drastic changes of traffic probably imply failures.

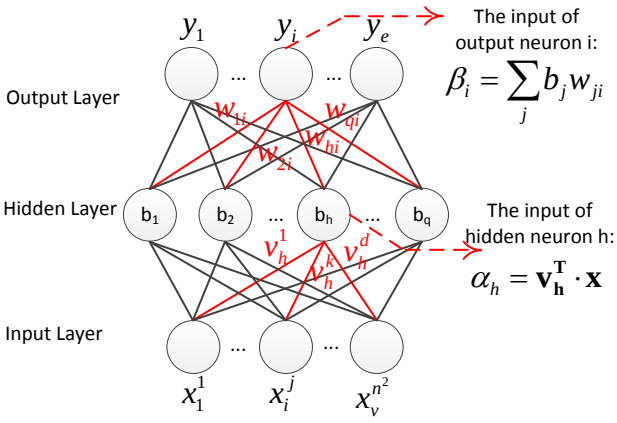

Figure 3: Multi-layer Perceptron Architecture

To collect required traffic data, we simply install *tcpdump* on each router and switch. These data will be collected to **Traffic Analysis Worker** for further analysis.

In order to model this data analysis problem, this paper proposes a Back Propagation Neural Networks (BPNN) based method to compute failure probability of each link.

Feature Extraction.

The first step is to extract rational features from raw traffic data. To address this issue, we extract $v \cdot n^2$ features in total. v is number of devices (CRs, ARs, ASs), each device has n^2 features where n is the number of racks. In particular, each feature denotes traffic of one direction. Obviously, there are n choices of sources and n choices of destinations, so the total number of possible traffic directions is n^2. Formally, the traffic features during a time period ΔT is presented as $\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_v})$, $\boldsymbol{x_i} = (x_i^1, x_i^2, ..., x_i^{n^2})$ where x_i^j stands for direction j traffic of device i in ΔT.

BPNN Model.

Suppose the number of links is e, for each time period ΔT, the output of our model should be $\boldsymbol{y} = (y_1, y_2, ..., y_e)$, where y_i denotes failure probability of link i. We apply a BPNN model to localize the faulty link during ΔT. As is shown in Figure 3, BPNN is a multi-layer perceptron architecture: To reduce the computational work of training, we only set one hidden layer, both the hidden layer neurons and output layer neurons utilize Sigmoid function as the activation function. Take the output layer as an example: $y_i = Sigmoid(\beta_i - \theta_i)$ where θ_i is a constant term of the perceptron.

Training of BPNN.

To train the neural network, we apply the classical error BackPropagation (BP) algorithm [11]. BP algorithm is based on gradient descent strategy. ?????

The Detection Algorithm.

BPNN model is only able to identify the faulty link during one time unit, in the following part, we will introduce how we can detect multiple faults over time.

First of all, we assume that there are at most one link failure during a time unit (ΔT). This assumption is partially supported by the finding proposed by Gill et. al. They find

that on an average, link failures tend to be separated by a period of about one week [6].

Algorithm 2 Traffic Analysis Worker Algorithm

Input: $\boldsymbol{x}^t = (\boldsymbol{x}_1^t, \boldsymbol{x}_2^t, ..., \boldsymbol{x}_v^t)$
Output: F_i for $i \in E$
1: divide traffic data $\boldsymbol{x}^t$ into $t/\Delta T$ chunks
2: **for** $i \in E$ **do**
3: $F_i \Leftarrow 0$
4: **end for**
5: **for** each data chunk $\boldsymbol{x}$ **do**
6: $\boldsymbol{y} \Leftarrow BPNN(\boldsymbol{x})$
7: **for** $i \in E$ **do**
8: $F_i \Leftarrow F_i + y_i$
9: **end for**
10: **end for**
11: **for** $i \in E$ **do**
12: $F_i \Leftarrow \frac{\Delta T F_i}{t}$
13: **end for**
14: sort $\boldsymbol{F}$ in descending order

Algorithm 2 shows the algorithm of **Traffic Analysis Worker**. The input of the algorithm is the refined traffic data over time T. Traffic data is then divided into chunks and each chunk represents traffic data of one time unit ΔT (line 1).

6. EVALUATION

In this section, we will present how we evaluate our approach in terms of efficiency and effectiveness. Experiments results will also be shown in this section.

6.1 Experiments Setup

7. CONCLUSIONS

This paper proposes a failure detector model for data center networks.

8. ACKNOWLEDGMENTS

This work is partially supported by the National High-Technology Research and Development Program of China (Grant No.2014AA01A301).

9. REFERENCES

[1] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 13–24. ACM, 2007.

[2] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 354–363. IEEE, 2002.

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

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

[5] P. Felber, X. Défago, R. Guerraoui, and P. Oser. Failure detectors as first class objects. In *Distributed Objects and Applications, 1999. Proceedings of the International Symposium on*, pages 132–141. IEEE, 1999.

[6] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361. ACM, 2011.

[7] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62. ACM, 2009.

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

[9] H. Herodotou, B. Ding, S. Balakrishnan, G. Outhred, and P. Fitter. Scalable near real-time failure localization of data center networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1689–1698. ACM, 2014.

[10] S. Kandula, D. Katabi, and J.-P. Vasseur. Shrink: A tool for failure diagnosis in ip networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 173–178. ACM, 2005.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[12] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer. A new adaptive accrual failure detector for dependable distributed systems. In *Proceedings of the 2007 ACM symposium on Applied computing*, pages 551–555. ACM, 2007.

[13] R. Van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Middlewareą́98*, pages 55–70. Springer, 1998.