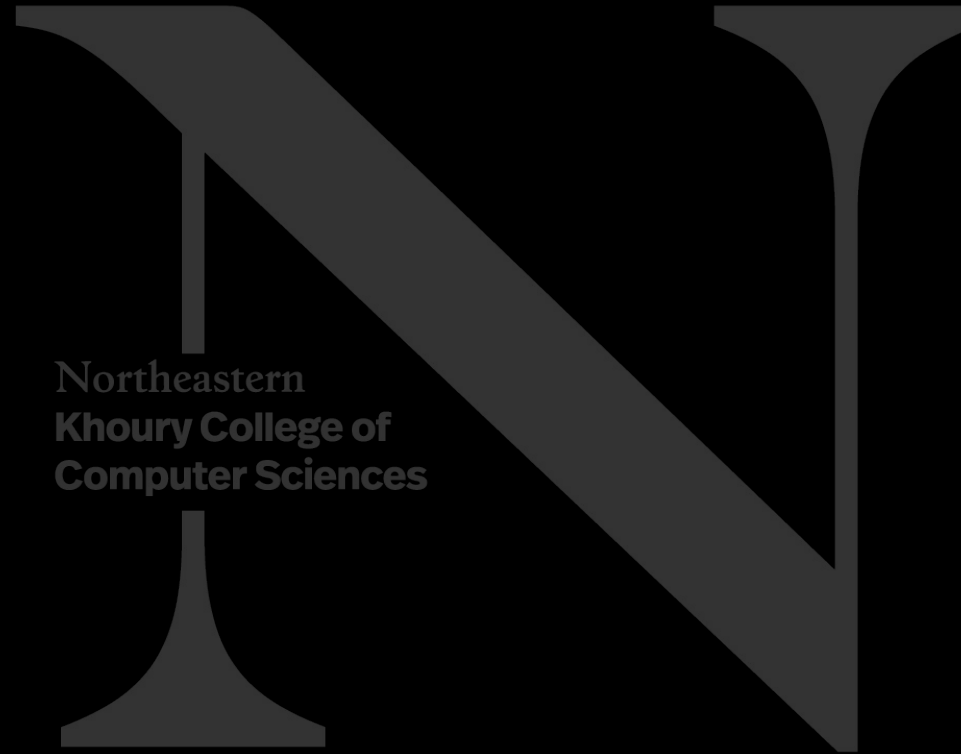


Study on High Availability & Fault Tolerance Application

Feb 20, 2023

Norman Kong Koon Kit and Michal Aibin



Edge Computing, Cloud Computing, and Big Data
International Conference on Computing, Networking and Communications (ICNC 2023)

Contents



Motivation



Problem Overview



Approach & Result



Practical Implementation



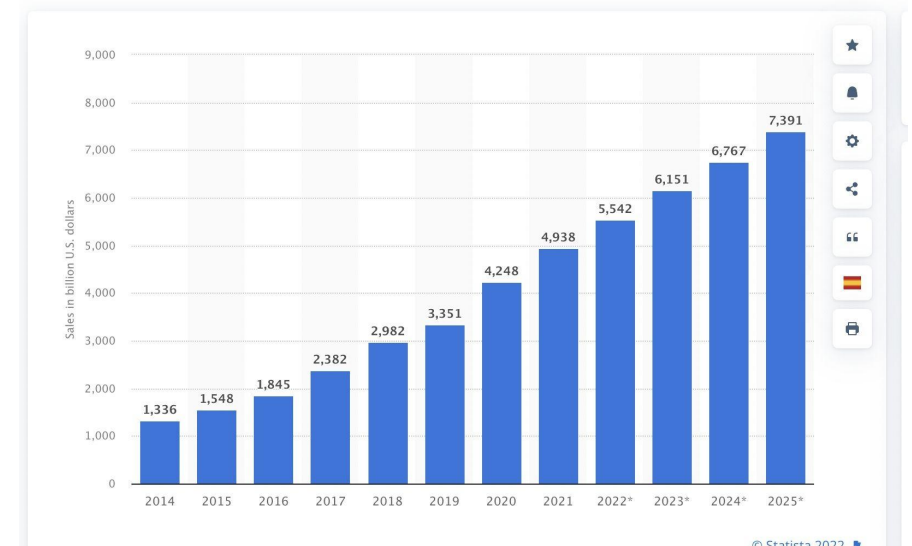
Conclusion

Motivation

Motivation

- E-commerce and Remote Working have been the new normal since the pandemic.
- e-Commerce trend grew 4 times from 1336 billions in 2014 and expected to be 7391 billions over 10 years
- Remote jobs have kept increasing from 30,000 to 60,000 from 2018 to 2021

Retail e-commerce sales worldwide from 2014 to 2025
(in billion U.S. dollars)



Motivation

NEWS


f in

Microsoft Teams Hit with a 6-Hour Outage

BY KURT MACKIE | JULY 21, 2022

Microsoft has confirmed that its Microsoft 365 apps and Teams services were down for some users on Wednesday night.

Update 7/22: Exoprise, a company that offers performance-monitoring solutions to detect service outages, estimated that the July 20 Microsoft Teams outage lasted three hours, per [this blog post](#).



BREAKING NEWS 7:03 36°

MAJOR FAA COMPUTER FAILURE
SYSTEM OUTAGE COULD GROUND FLIGHTS NATIONWIDE

GMA
GOODMORNING AMERICA.COM

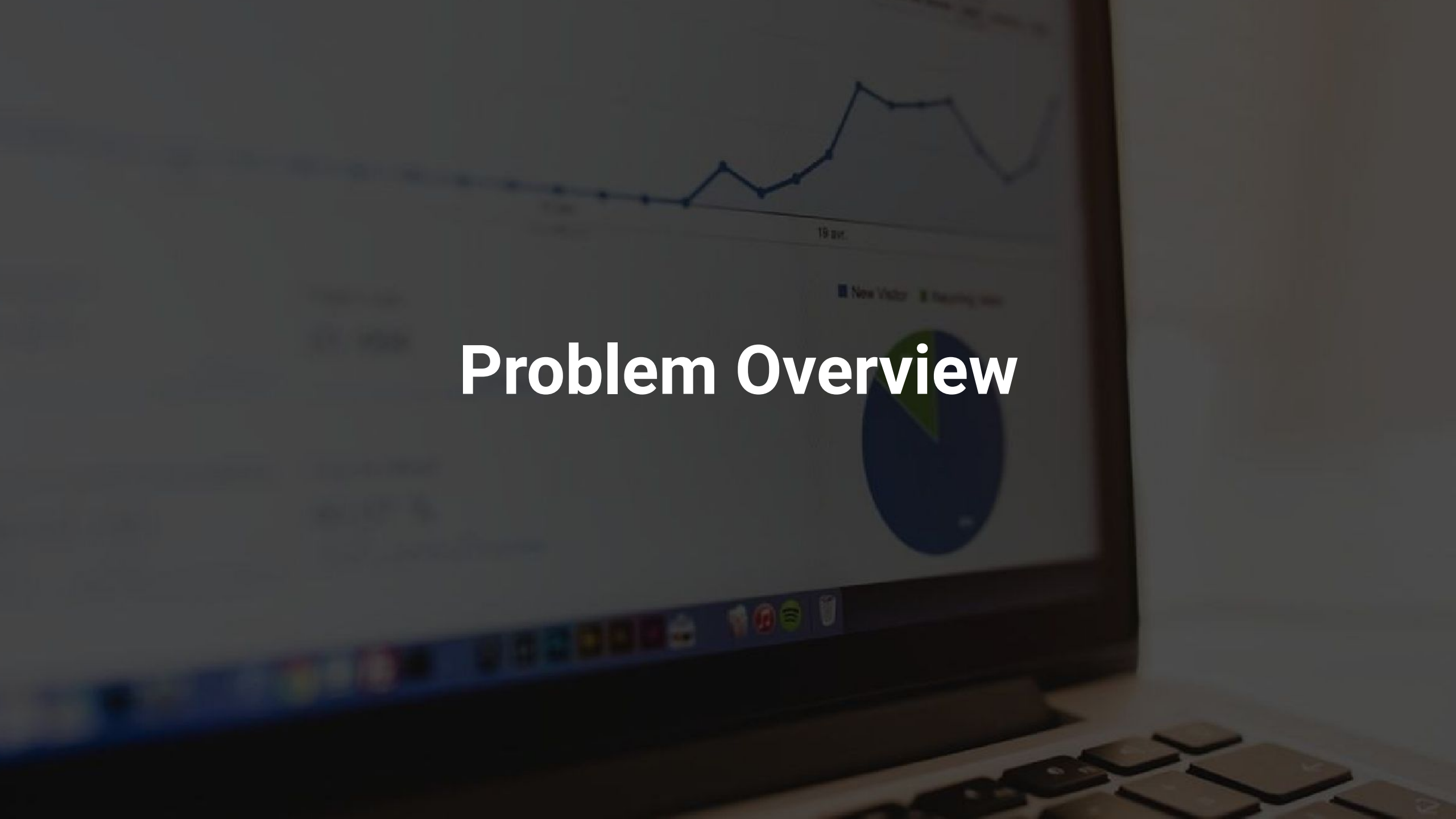
Y TO UNDERGO HIP SURGERY THIS WEEK • TEXAS ANNOUNCES RECORD \$33B BUDGET SURPLUS

As more or more critical system like Online Payment, Air Traffic Control are put online.

It is crucial to design application with **High Availability and Fault Tolerance**, otherwise the consequence will be disastrous...

In this research, we would like to perform deep dive into application design to improve the overall Availability and Fault tolerance

Problem Overview



Problem Overview

The term “Availability” was defined as “The degree to which a system is functioning and is accessible to deliver its services during a given time interval”

⇒ Maximum downtime percentage over a given period

Years of continuous operations	1	2	3
Availability	Maximum allowable downtime		
99,0000% (2–9s)	3 d 15 h 36 min 0 s	7 d 7 h 12 min 0 s	10 d 22 h 48 min 0 s
99,9000% (3–9s)	8 h 45 min 15 s	17 h 31 min 12 s	1 d 2 h 16 min 48 s
99,9900% (4–9s)	52 min 34 s	1 h 45 min 7 s	2 h 37 min 41 s
99,9990% (5–9s)	5 min 15 s	10 min 31 s	15 min 46 s
99,9999% (6–9s)	32 s	1 min 3 s	1 min 35 s

} High Availability

Source : Service Availability: Principles and Practice (Maria Toeroe · Francis Tam 2012)

Problem Overview

- Availability is a measure of the percentage of time that an application is running properly, i.e.

$$Availability = \frac{uptime}{uptime + downtime} * 100\%$$

To further breakdown the formula, we define uptime & downtime as :

Problem Overview

$$Availability = \frac{uptime}{uptime + downtime} * 100\%$$

$$Availability = \frac{MTBF}{MTBF + MTTR} * 100\%$$

MTBF = Mean Time Between Failure

MTTR = Mean Time To Restore/Recovery

To improve, we can either to \uparrow MTBF or \downarrow MTTR

In this research, we will focus on \downarrow MTTR as much as possible

$$Availability = \frac{uptime}{uptime + downtime} * 100\%$$

$$Availability = \frac{MTBF}{MTBF + MTTR} * 100\%$$

Problem Overview

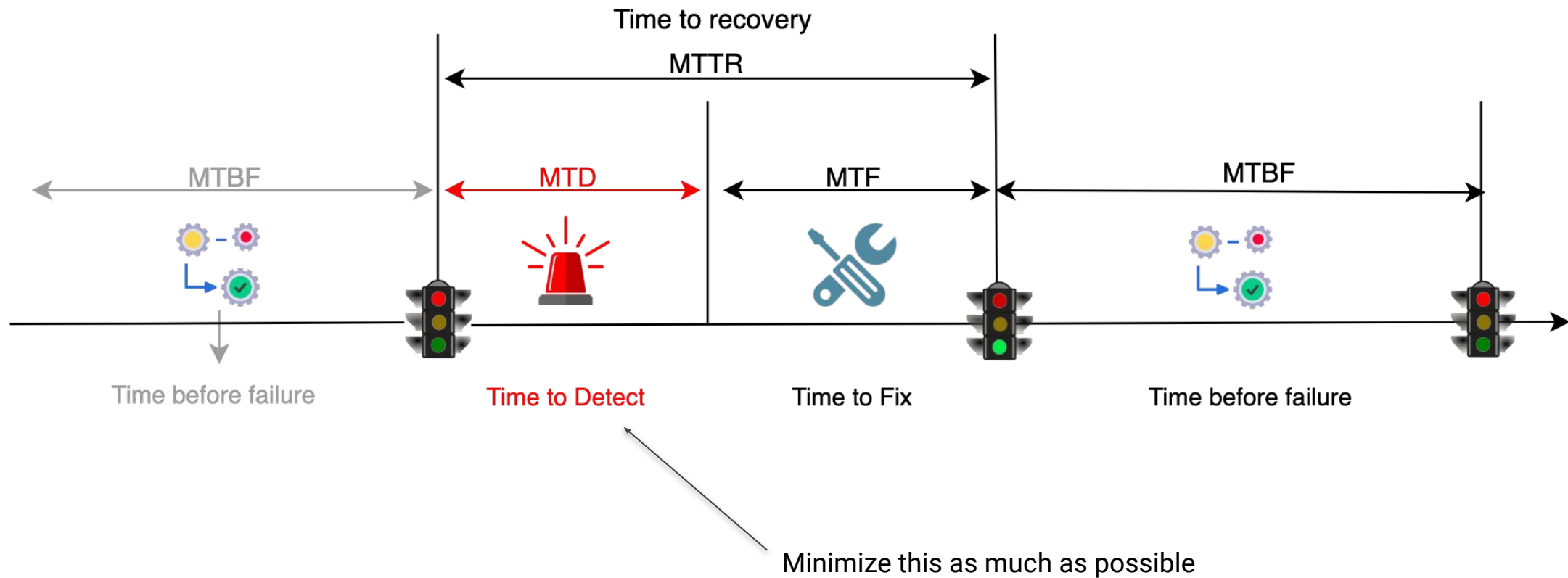
MTTR(Mean Time to Restore) can be further sub-classified into :

- **Mean Time to Detect** the failure
- **Mean Time to Fix** the failure

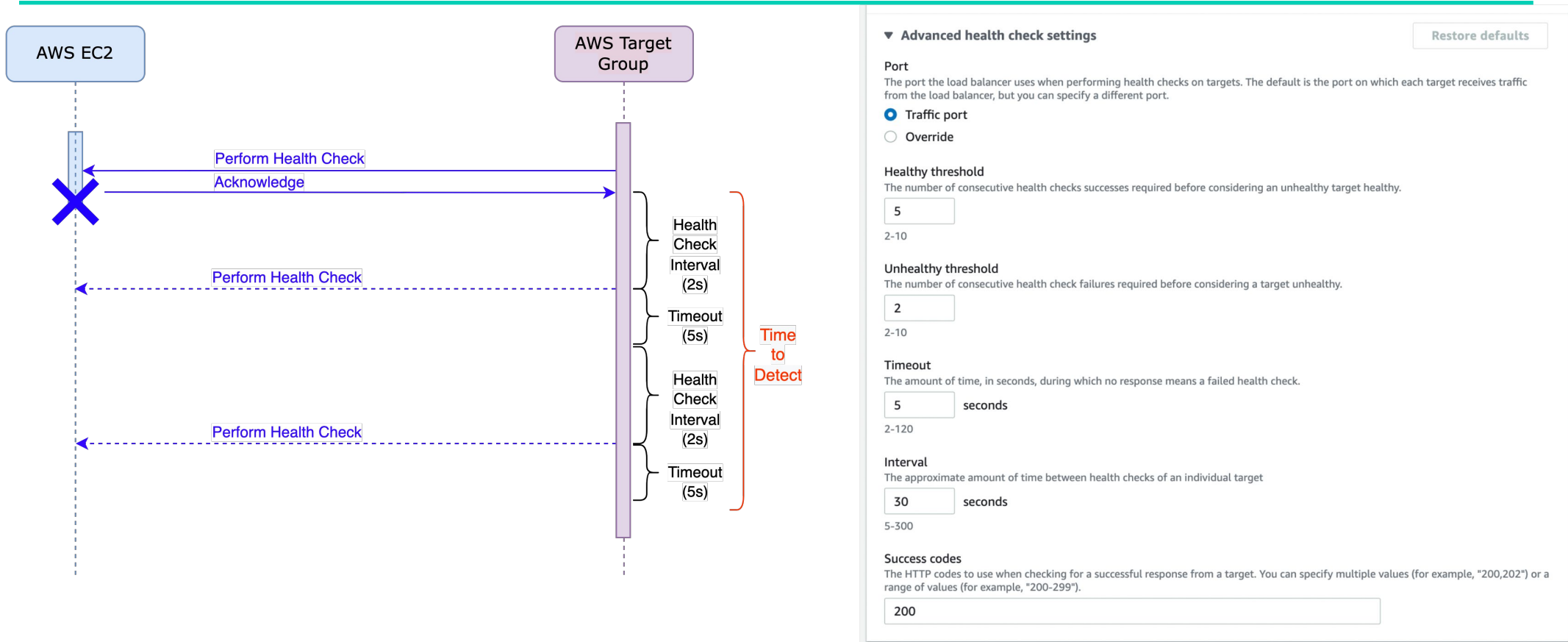
$$Availability = \frac{MTBF}{MTBF + MTD + MTF} * 100\%$$

This research will focus on Minimize the **MTD (Mean Time Detection)**

Problem Overview



Industrial Practice



To ensure the underlying application is up and running, typical Load Balancer will perform a **“Pull-based”** health check (or probe) periodically, which also include a timeout and Retry before declaring the node is unhealthy, worst case up to **14 seconds in AWS**

How can we improve it ?

The Approach



The Approach

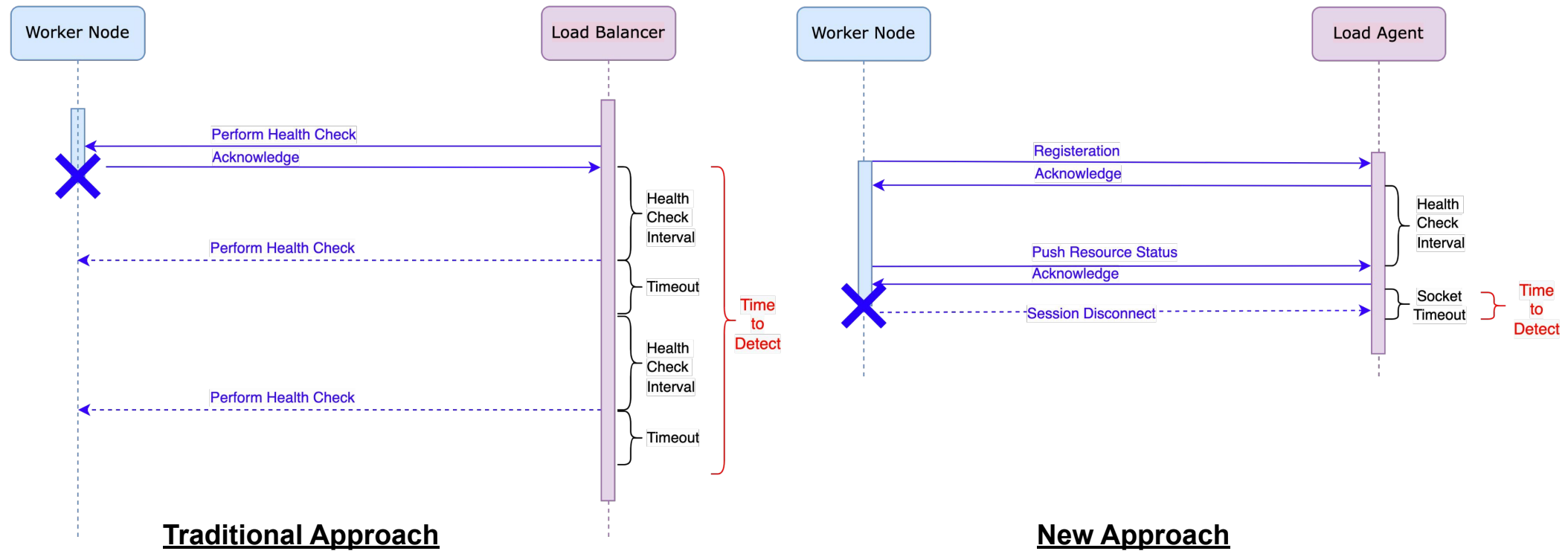
Instead of “Pull Based” health check, we propose a pluggable “Resource Agent”

- Push Based health check
- Persistent Connection

- 1) Agent keep a **persistent connection** to the “*Load Agent*”
- 2) Node information (CPU / Memory / Network IO) will be sent periodically
- 3) “*Load Balancer*” perform a “*Resource Awareness Routing Algorithm*” to dispatch client request.

Note : When worker node **crashes**, this “*Persistent Connection*” will be lost such that “*Load Agent*” update the Load Balancer with **minimal delay**.





Minimize "Time to Detect"



Instead of Pull-based health check, we will use Push Based + Persistent Connection to improve the Time for Detection

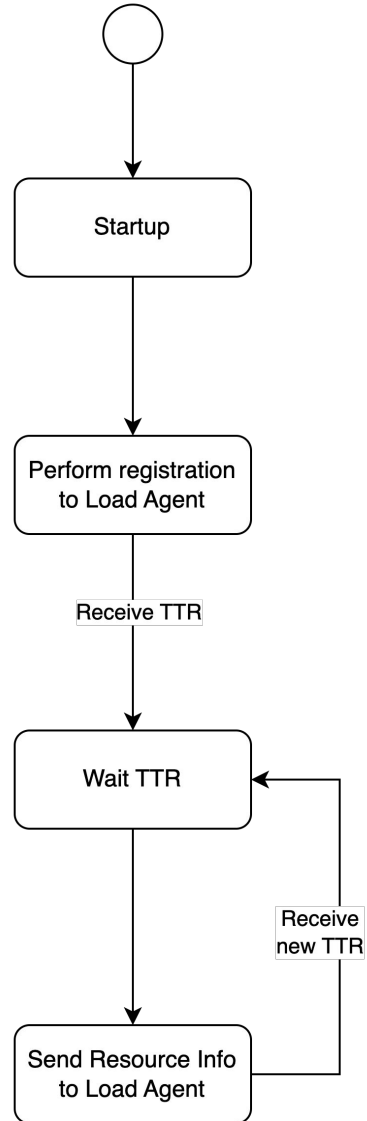
Components in this Proposal

There are 4 major components involves :

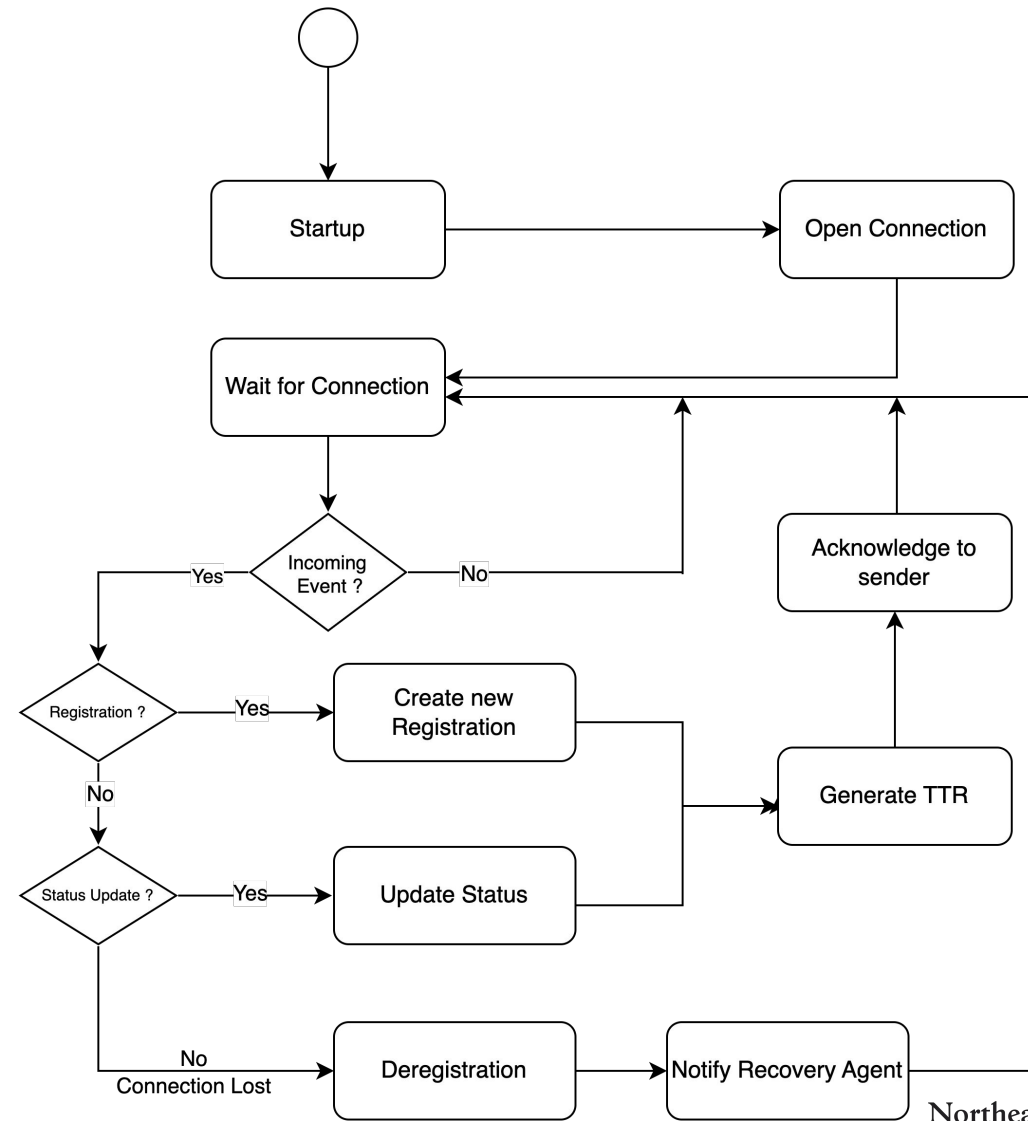
- 1) *Load Agent* 
Wait for worker node connect and update node information to cache
- 2) *Resource Agent* (Embedded in Worker Node) 
Establish a persistent connection to Load Agent and push status periodically
- 3) *Load Balancer* 
Run “Resource Awareness Routing Algorithm” to dispatch to Worker Node
- 4) *Recovery Agent* 
Perform recovery action upon Load Agent detect outage

Note : Recovery Agent is to minimize the MTF(Mean Time to Fix) instead of MTD

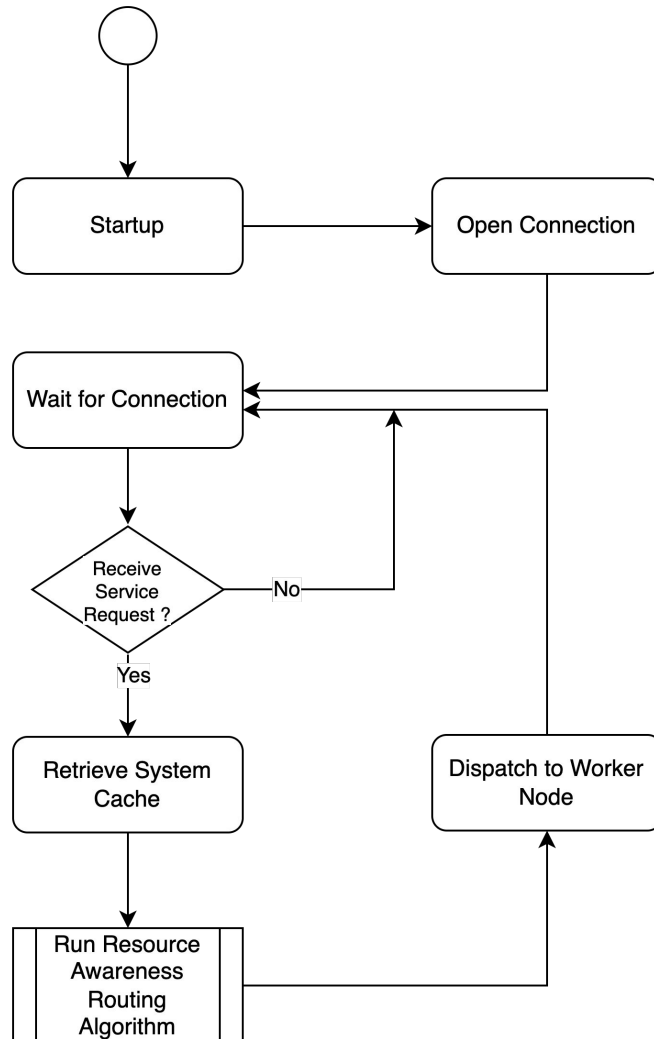
Resource Agent



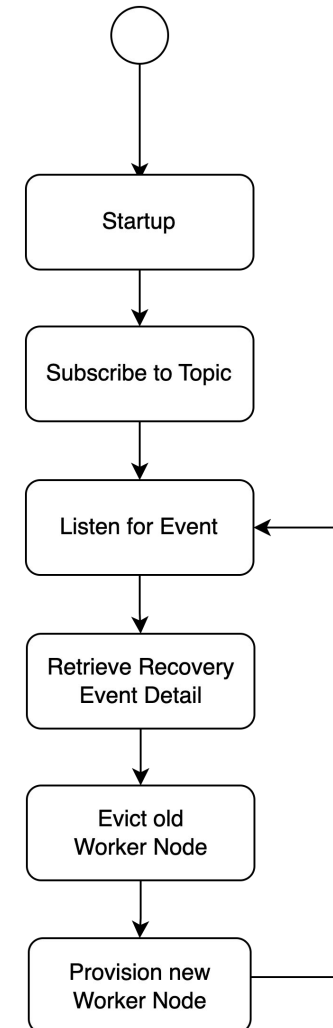
Load Agent



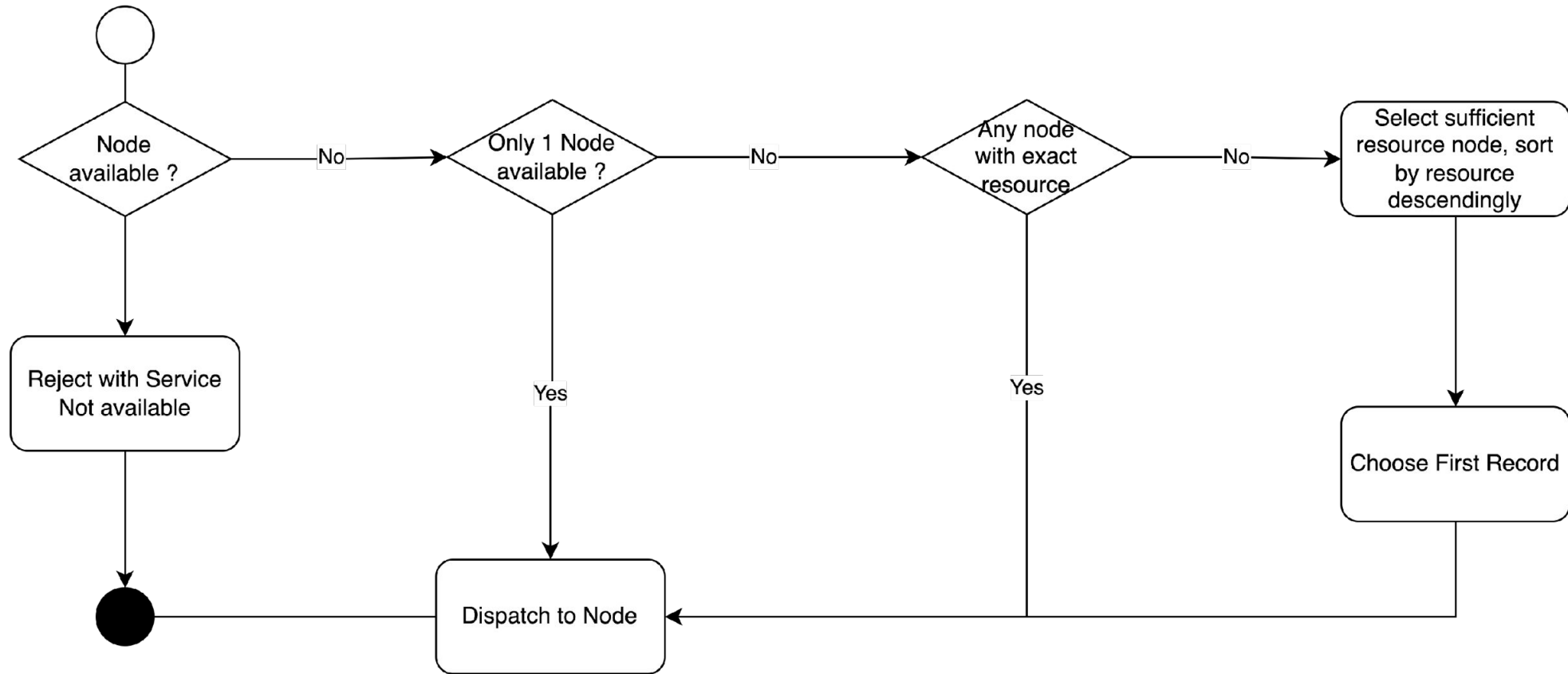
Load Balancer

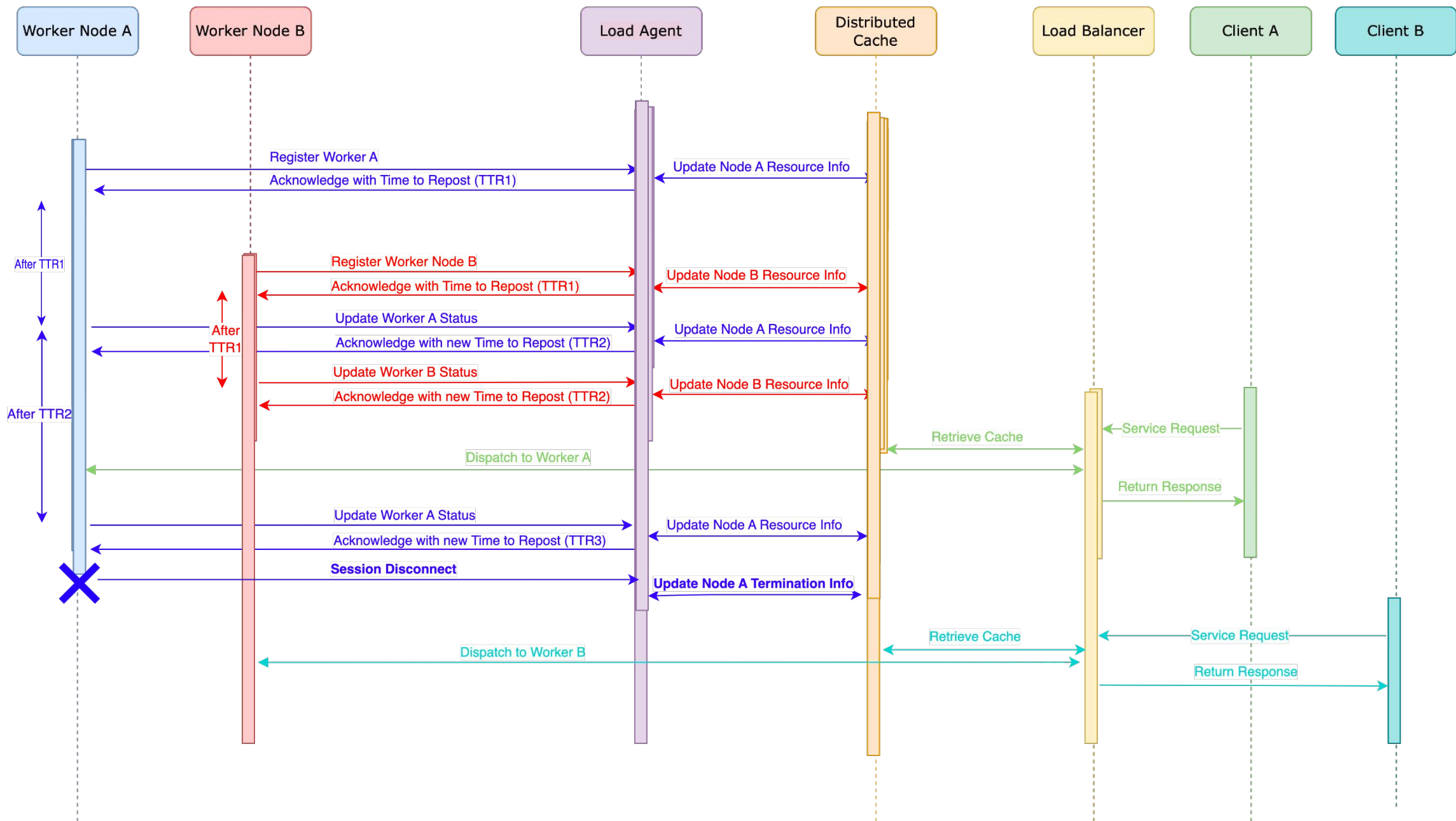


Recovery Agent



Resource Awareness Routing Algorithm





Our Sample Implementation

<input type="checkbox"/>	load_agent 7dec18f9d00e (LoadAgent)	less than a minute ago	running	
<input type="checkbox"/>	load_balancer 21cea46e6486 (LoadBalancer)	1 minute ago	running	
<input type="checkbox"/>	redis 0a1185277266 (redis)	1 minute ago	running	
<input type="checkbox"/>	resource_agent 4141aeaf68c0 (WorkerNode--20220703_011753-25895574)	1 minute ago	running	

- Applications are implemented in NodeJS with WebSocket as the persistent connection
- Redis is used as the Memory Datastore and Message Queue Broker
- Components are deployed as Docker, except Recovery Agent
- Recovery Agent executes docker command to perform recovery action

The Result



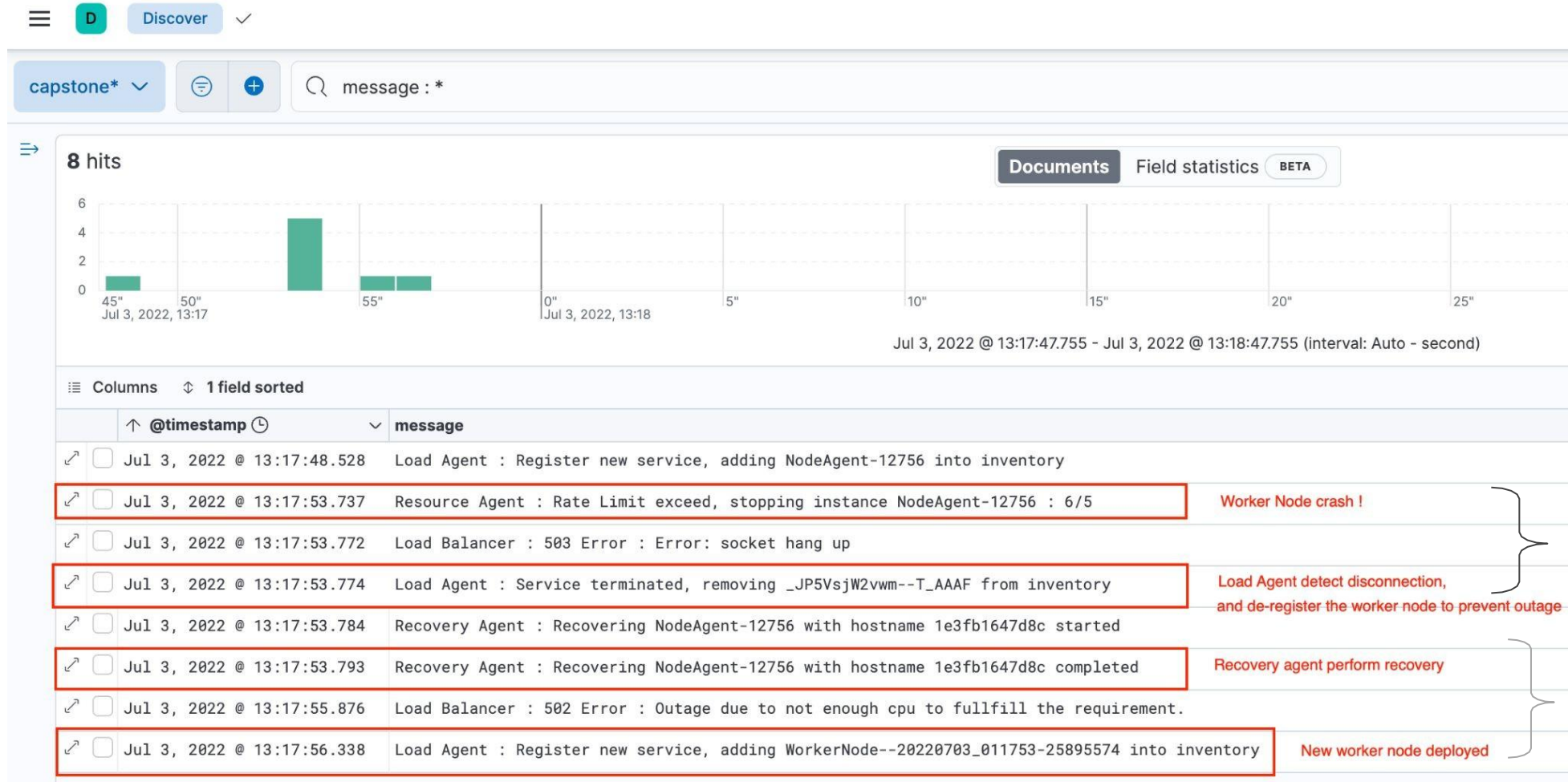
Test Result 1 - Log Analysis

In our proposed scenario, we run 1,000 system crash simulation and the average “Time to detect” is

24.5 milliseconds

Recall AWS worst case scenario is 14 seconds, which is 583 times faster !

Test Result 1 - Log Analysis



MTD ~
37ms

MTF ~ 2.8s

Test Result 2 - Compare overall SLA

- 1) Deployed similar application to AWS EC2
- 2) Compare with the proposed framework

In order to simulate the system crash, a “Kill Switch” is implemented to "crash" after pre-defined duration :

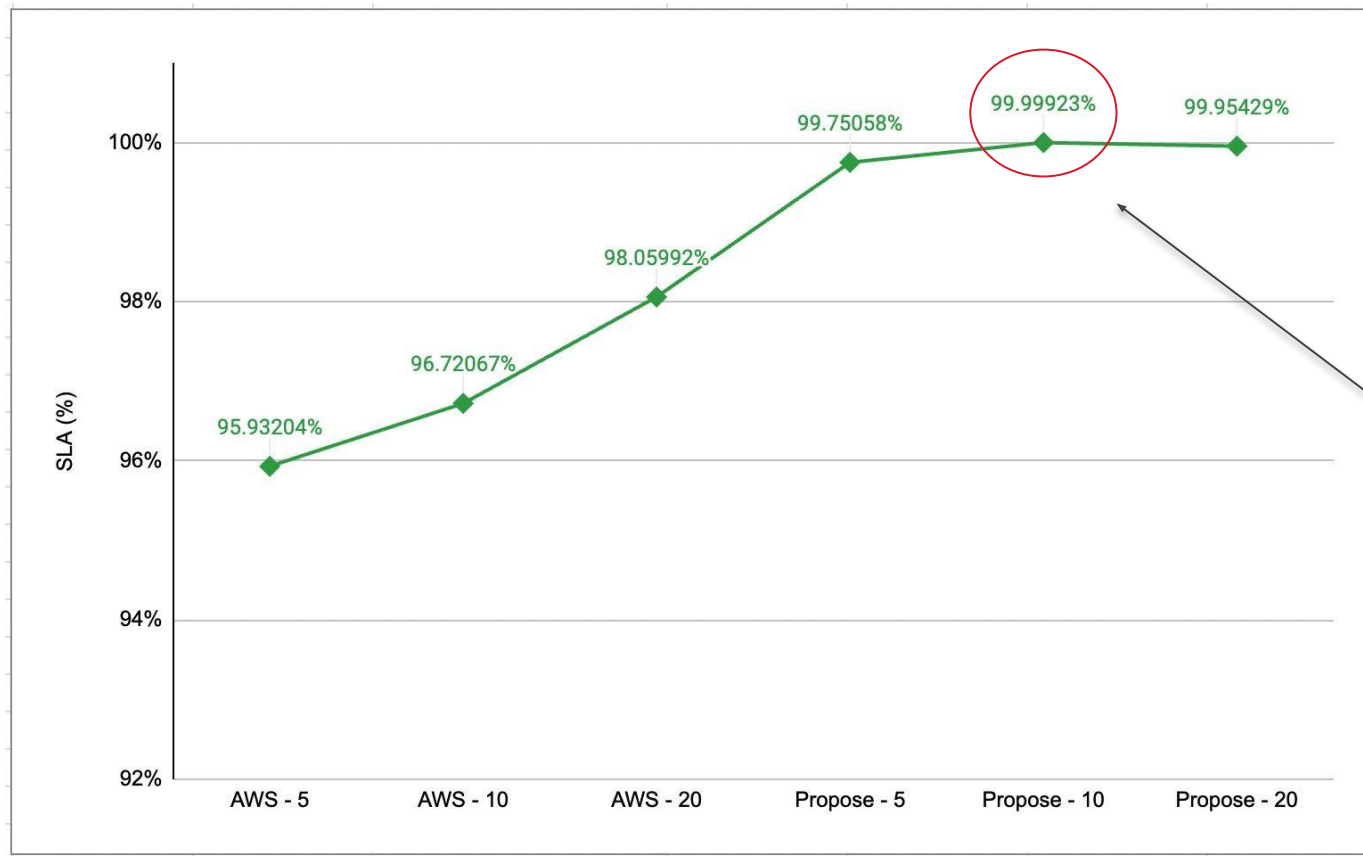
#	Kill Switch Frequency
1	5 minutes of execution
2	10 minutes of execution
3	20 minutes of execution

Test Result 2 - Compare overall SLA

	Duration(s)	Up Time(s)	Downtime(s)	SLA(%)	Fail Count
AWS - 5 min	2105.901	2020.234	85.667	95.93204%	2,080
AWS - 10 min	2203.67	2131.404	72.266	96.72067%	1,518
AWS - 20 min	2089.374	2048.838	40.536	98.05992%	1,021
Propose - 5 min	2943.944	2936.602	7.343	99.75058%	3,990
Propose - 10 min	2970.906	2970.883	0.023	99.99923%	600
Propose - 20 min	3005.539	3004.165	1.374	99.95429%	308

Disclaimer : AWS Load Balancer is on Virtual Machine while the simulation is riding on Docker.
The recovery time for VM is much higher than docker

Test Result 2 - Compare overall SLA



Meet High Availability SLA !

Years of continuous operations	1	2	3
Availability	Maximum allowable downtime		
99.0000% (2-9s)	3 d 15 h 36 min 0 s	7 d 7 h 12 min 0 s	10 d 22 h 48 min 0 s
99.9000% (3-9s)	8 h 45 min 15 s	17 h 31 min 12 s	1 d 2 h 16 min 48 s
99.9900% (4-9s)	52 min 34 s	1 h 45 min 7 s	2 h 37 min 41 s
99.9990% (5-9s)	5 min 15 s	10 min 31 s	15 min 46 s
99.9999% (6-9s)	32 s	1 min 3 s	1 min 35 s

Practical Implementation



Practical Implementation

Since the push-based + persistent mechanism is CPU resource consuming, this framework is suitable for **Mission Critical** applications like :

- 1) Security trading system
- 2) Banking system
- 3) Air traffic control system

Conclusion



Conclusion

- High availability has been one of the biggest challenges in application design
- Depends on the use cases, there are various techniques can improve the service availability
- This research paper proposes a “Push-based mechanism with persistent connection” to reduce the “Time to Detect” such that the overall Service Level Agreement can be improved