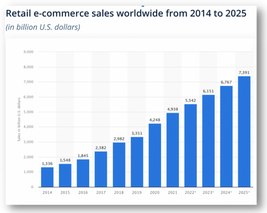
Northeastern University
Khoury College of Computer Sciences

NORMAN KONG KOON KIT
MICHAL AIBIN

STUDY ON HIGH AVAILABILITY & FAULT TOLERANCE APPLICATION

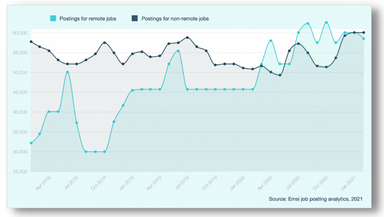
Background



E-commerce and remote working have been the new normal since the pandemic.

e-Commerce trend grows 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



It is crucial to design application with High Availability and Fault Tolerance

High availability has been one of the biggest challenges in application design

This research proposes a "Push-based mechanism with persistent connection" to reduce the "Time to Detect" such that the overall *Service Level Agreement* can be improved

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

The Definition of Service Level Agreement based on Availability is :

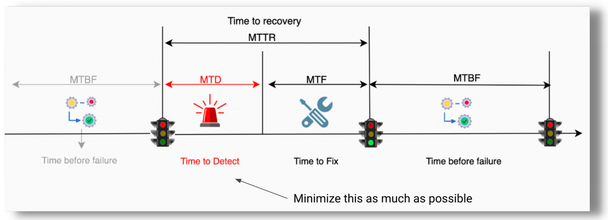
$$Availability = \frac{uptime}{uptime + downtime} * 100\%$$

We can further breakdown the calculation as below components :

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

where
MTBF Mean to Before Failure
MTD Mean Time to Detect
MTF Mean Time to Fix

Graphically,



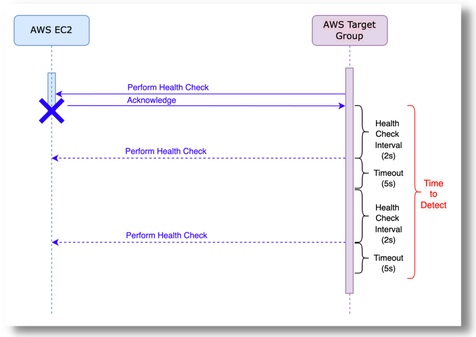
This research is to minimize the MTD in order to improve the overall Service Level Agreement

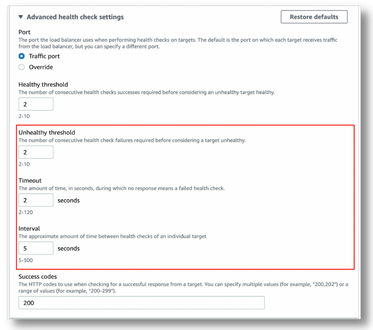
Approach

Traditionally, most of the fault detection is using "Pull" based approach, i.e. a centralized agent periodically send Health Check requests to query the status of underlying worker node.

This approach is the most common implementation with pretty good performance

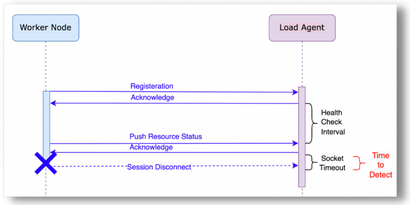
However, in the worst case scenario, the "Time to Detect" will be 14s (when comparing with AWS Load Balancer)





We propose a pluggable "*Resource Agent*" that embedded into worker node, this agent keep a **persistent connection** to the "*Load Agent*". By **periodically** updating the resource information (CPU / Memory / Network IO), "*Load Balancer*" perform perform a "*Resource Awareness Routing Algorithm*" to dispatch request.

When worker node **crashes**, the "*Persistent Connection*" will be disconnected such that "*Load Agent*" can detect and update the Load Balancer with **minimal delay**.



There are 4 majors components in this middleware :

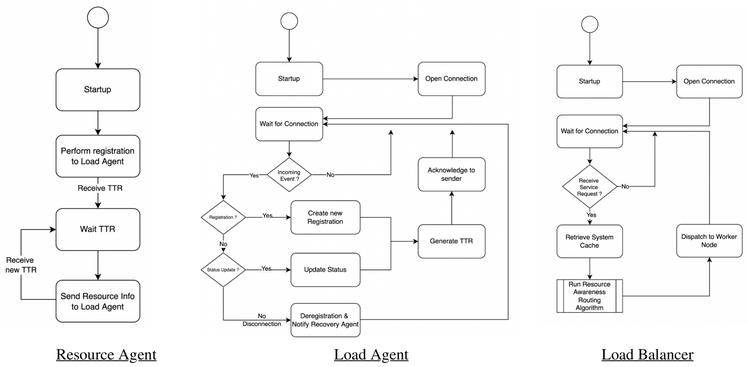
1) *Load Agent*
Wait for worker node connect and update node information to cache

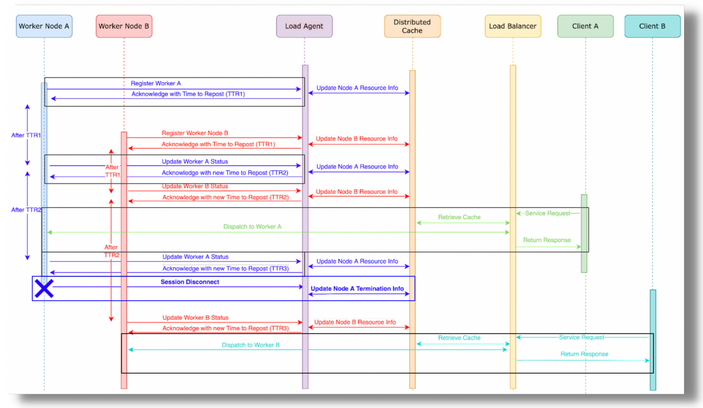
2) *Resource Agent* (Embedded in Worker Node)
Establish a persistent connection to Load Agent and push status periodically

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





When Worker Node A is down, session disconnection event will trigger Load Agent to remove from resource pool, such that Load Balancer will not dispatch request to this problematic node

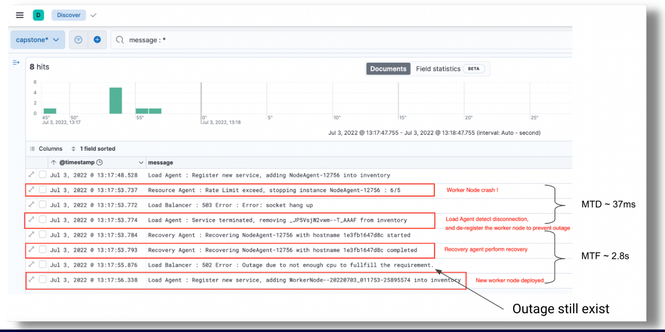
Result

We have implement a sample application using NodeJS = Socket.io with two testing approaches :

1) Trigger 1,000 system crashes, then collect the metrics for analysis

Result : Average "Time to Detect" is 24.5ms, which is 583 times faster than the theoretical worst case value of AWS Load Balancer.

Below is one of the sample logs to illustrate the calculation :



2) Compare the overall Service Level Agreement with AWS Load Balancer

- Deploy the sample application on AWS EC2, with target group monitoring the EC2 HTTP status

- Apply different kill switch setting and calculate the overall Service Level Agreement against our proposed solution

The result indicated that our proposed solution is better than the AWS classic solution, while our 10 minutes kill switch even can achieve 5-nines (i.e. High Availability)

	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