AID: Efficient Prediction of Aggregated Intensity of Dependency in Large-scale Cloud Systems

Tianyi Yang, Jiacheng Shen, Yuxin Su, Xiao Ling, Yongqiang Yang, Michael Lyu

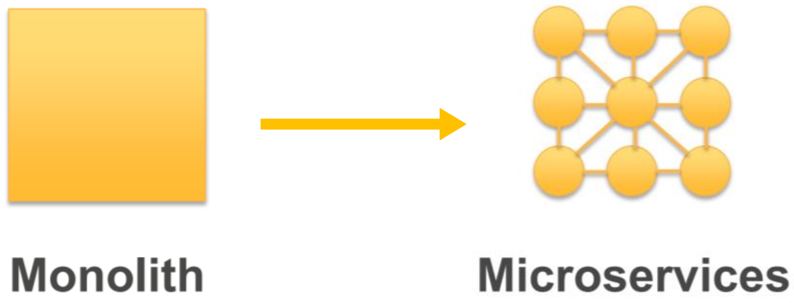
✉ tyyang@cse.cuhk.edu.hk

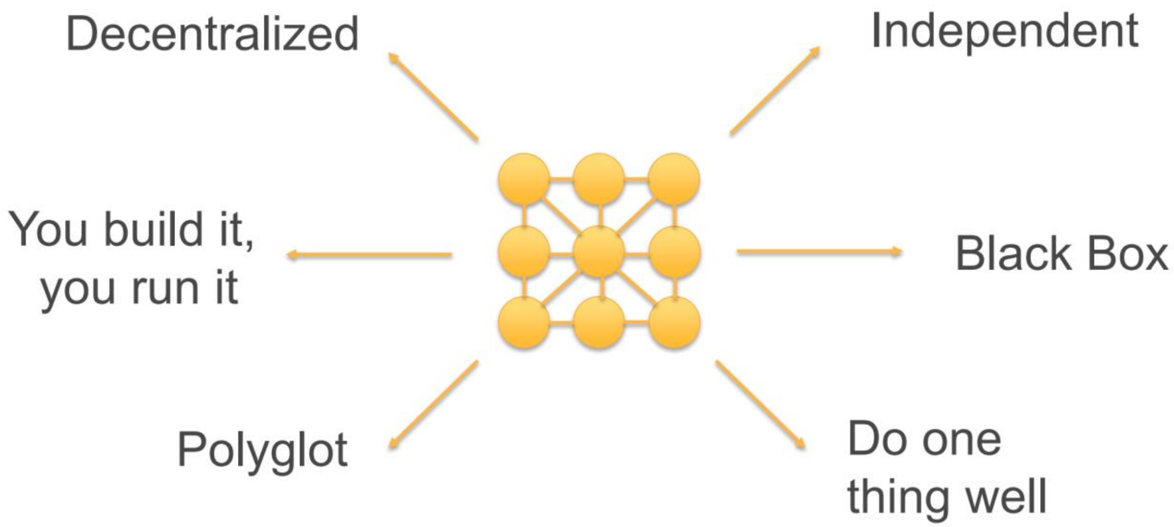
ARISE
Automated Reliable Intelligent Software Engineering

香港中文大學
The Chinese University of Hong Kong

Outline

The Microservices Architecture





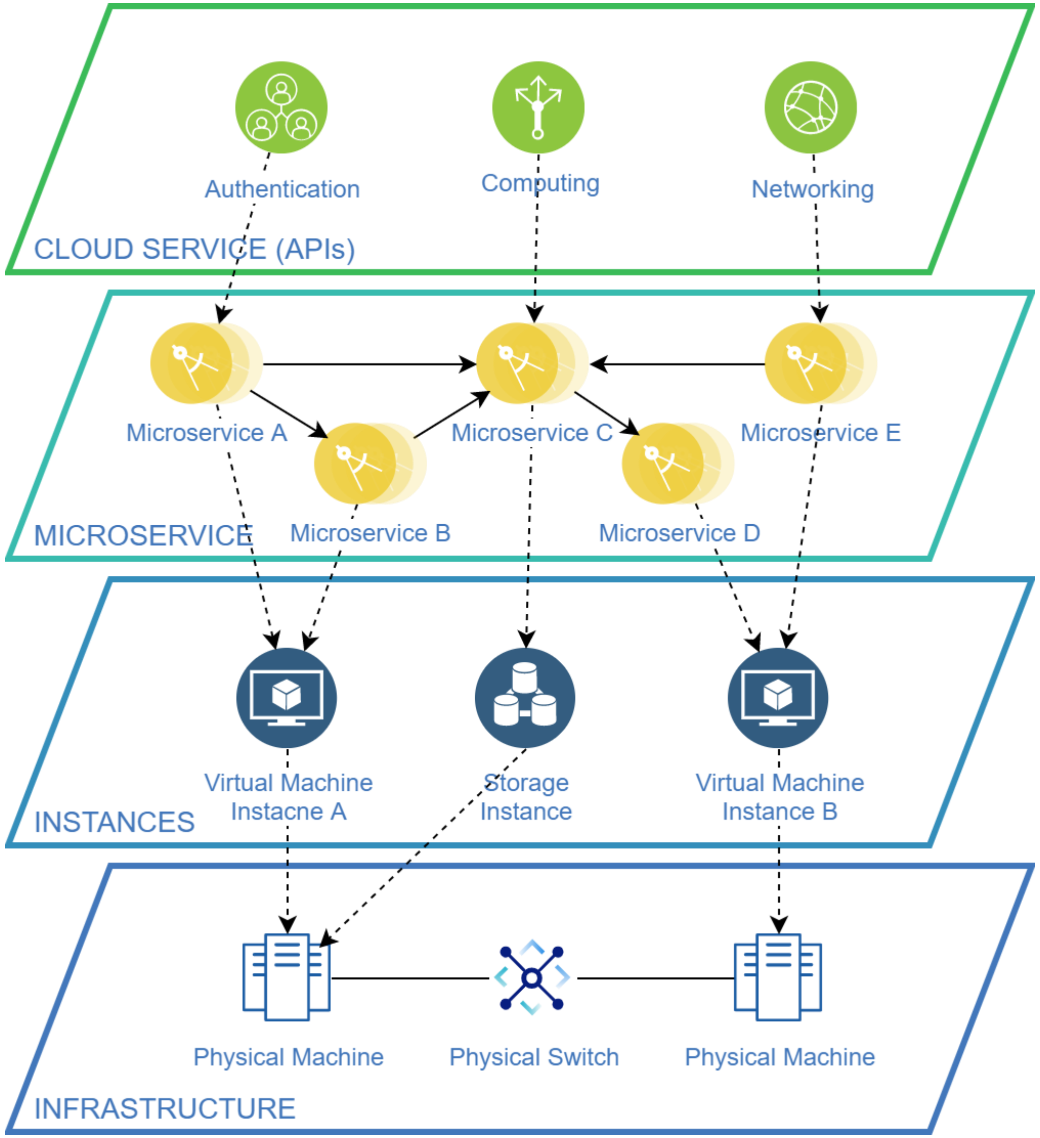
Microservices architecture is an approach in which a single application is composed of many **loosely coupled** and **independently deployable** smaller services.

Microservices on AWS, AWS Summit Berlin 2016, Apr 12, 2016
What are Microservices? | IBM

The Architecture of Cloud Systems

- Cloud microservices collectively comprise multiple cloud services.
  - <u>*Cloud services*</u>: provide high-level APIs.
  - <u>*Cloud microservices*</u>: collectively handle the external request via multiple chained invocations.
- Minor anomalies may magnify impact and escalate into system outages!

Loosely-coupled nature makes failure diagnosis difficult.

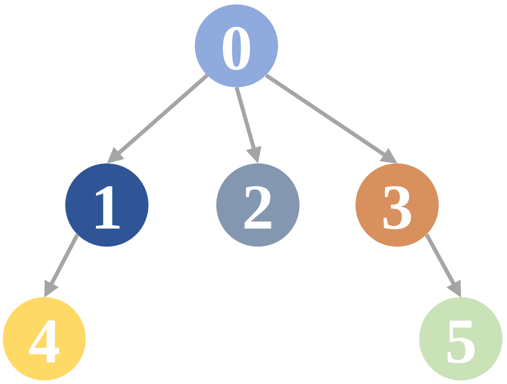


Distributed Tracing

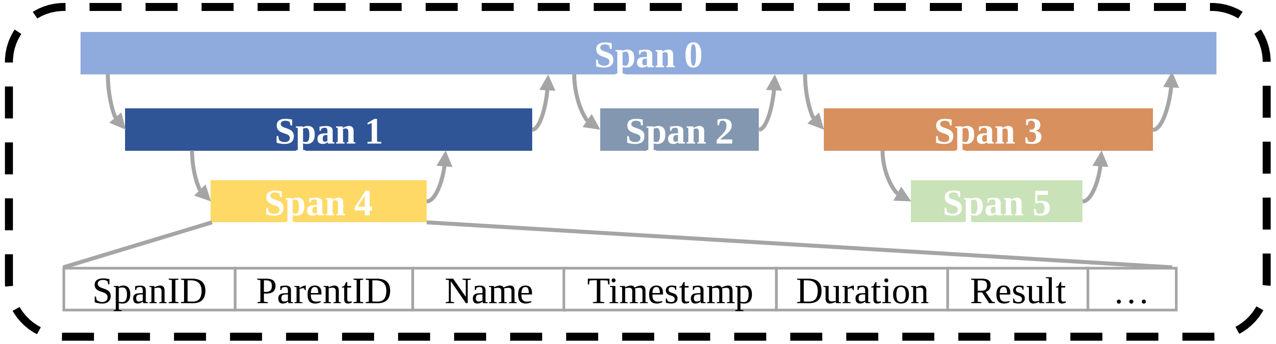
- Tracks the execution path of each request.
- Terminologies
  - *Span log (abbr. span)*: a log recording the contextual information of each service invocation.
  - *Trace log (abbr. trace)*: all the spans that serve for the same request.

Span ID	e22f30bdbfd09134
Parent Span ID	b42a04bf18997d5d
Name	ts-preserve-service
Timestamp (μs)	1618589098705000
Duration (μs)	1126
Result	SUCCESS
Trace ID	c0d17d481f47bdd9
Additional Logs	……

A span generated by the train-ticket benchmark.



Service invocations for a request.



A trace with 6 spans.

Outline

1. Background
2. Motivation
3. Methodology
4. Experiment
5. Case Study

A Survey of the Outages in AWS

AWS Post-Event Summaries

AWS Post-Event Summaries

The following is a list of post-event summaries from major service events that impacted AWS service availability:

- Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region, November, 25th 2020
- Summary of the Amazon EC2 and Amazon EBS Service Event in the Tokyo (AP-NORTHEAST-1) Region, August 23, 2019
- Summary of the Amazon EC2 DNS Resolution Issues in the Asia Pacific (Seoul) Region (AP-NORTHEAST-2), November 24, 2018.
- Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region, February 28, 2017.
- Summary of the AWS Service Event in the Sydney Region, June 8, 2016.
- Summary of the Amazon DynamoDB Service Disruption and Related Impacts in the US-East Region, September 20, 2015.
- Summary of the Amazon EC2, Amazon EBS, and Amazon RDS Service Event in the EU West Region, August 7, 2014.
- Summary of the Amazon SimpleDB Service Disruption, June 13, 2014.
- Summary of the December 17th event in the South America Region (SA-EAST-1), December 20, 2013.
- Summary of the December 24, 2012 Amazon ELB Service Event in the US-East Region, December 24, 2012.
- Summary of the October 22, 2012 AWS Service Event in the US-East Region, October 22, 2012.
- Summary of the AWS Service Event in the US East Region, July 2, 2012.
- Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region, April 29, 2011.

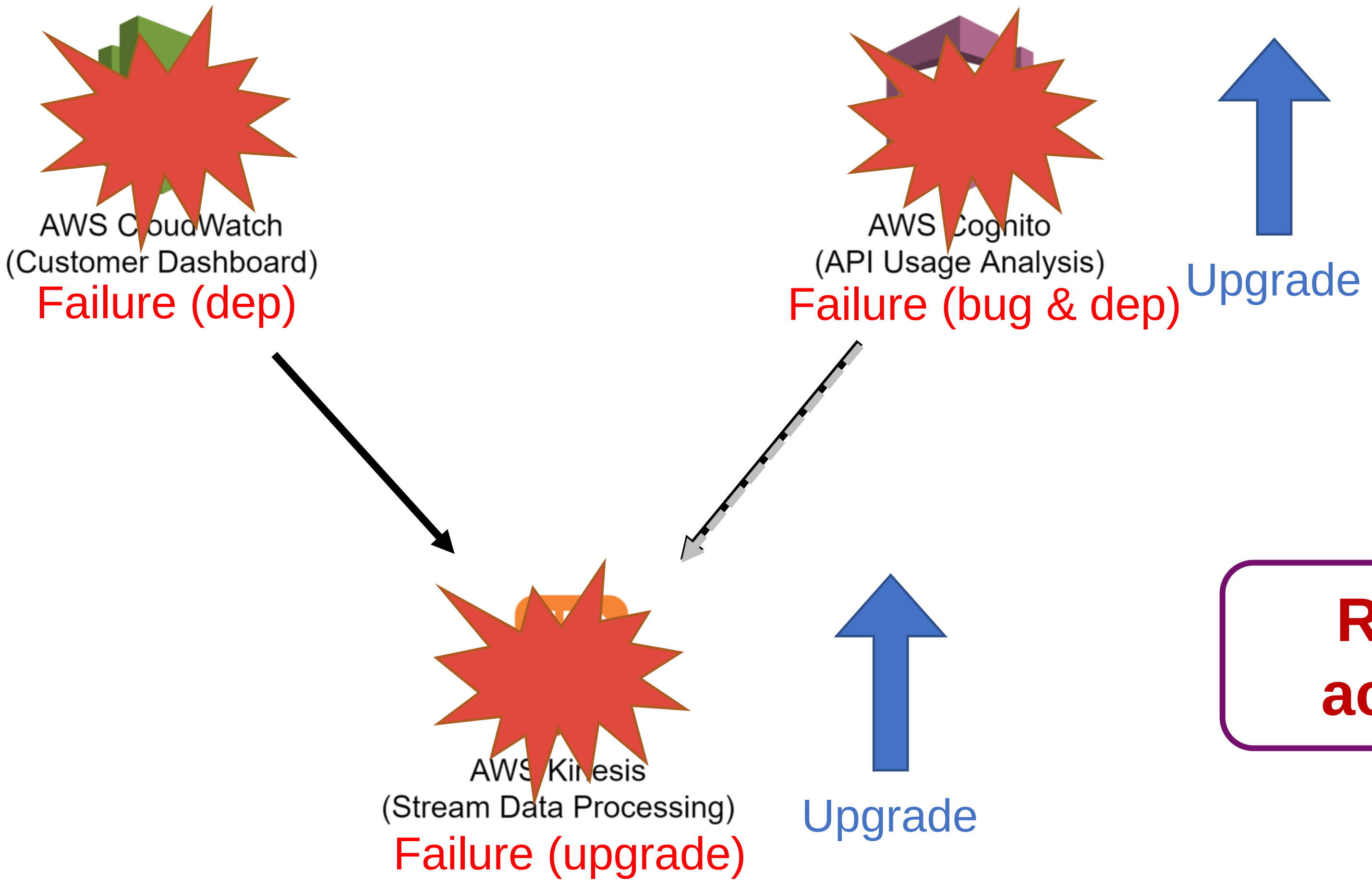
Slow Recovery

Cascading failure

5 out of 13 AWS outages are related to service dependency!

AWS Post-Event Summaries

AWS Kinesis Event on Nov 25th, 2020

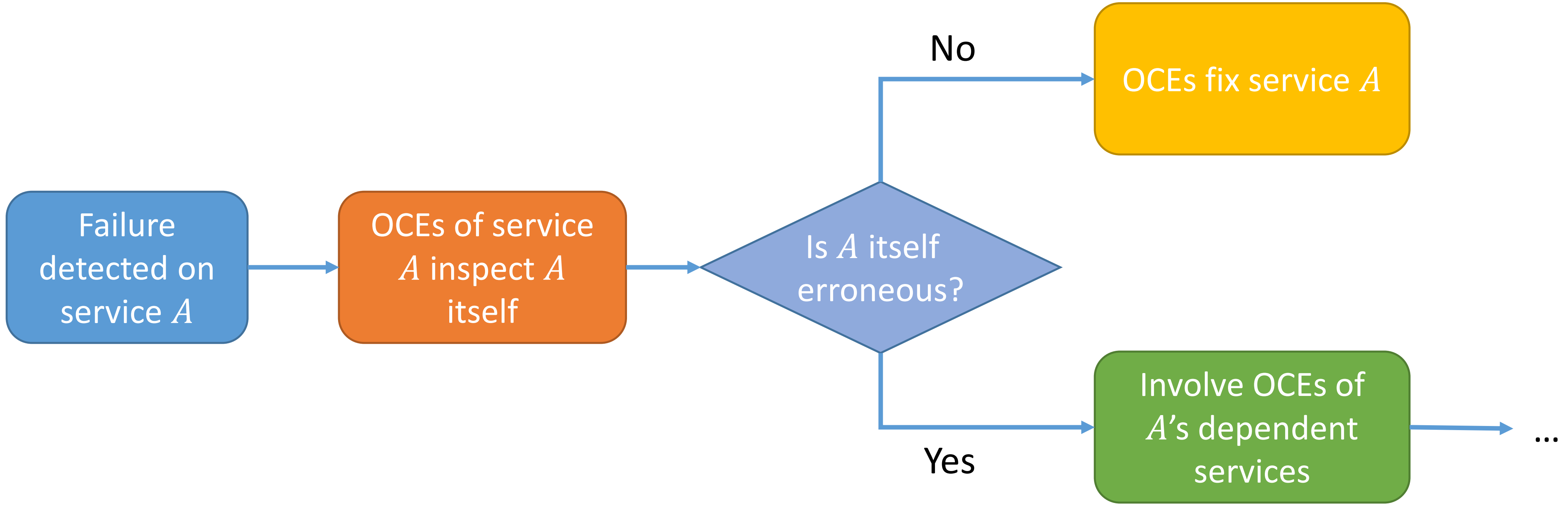


Reduced dependency can accelerate failure recovery.

[Northern Virginia (US-EAST-1) Region]

[Summary of the Amazon Kinesis Event in the Northern Virginia (US-EAST-1) Region]

Drawbacks of Current Failure Diagnosis Methods

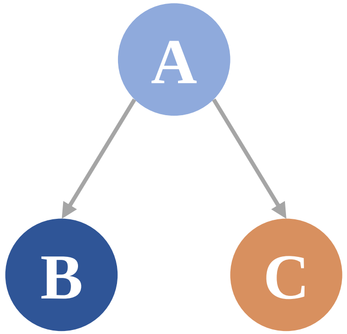


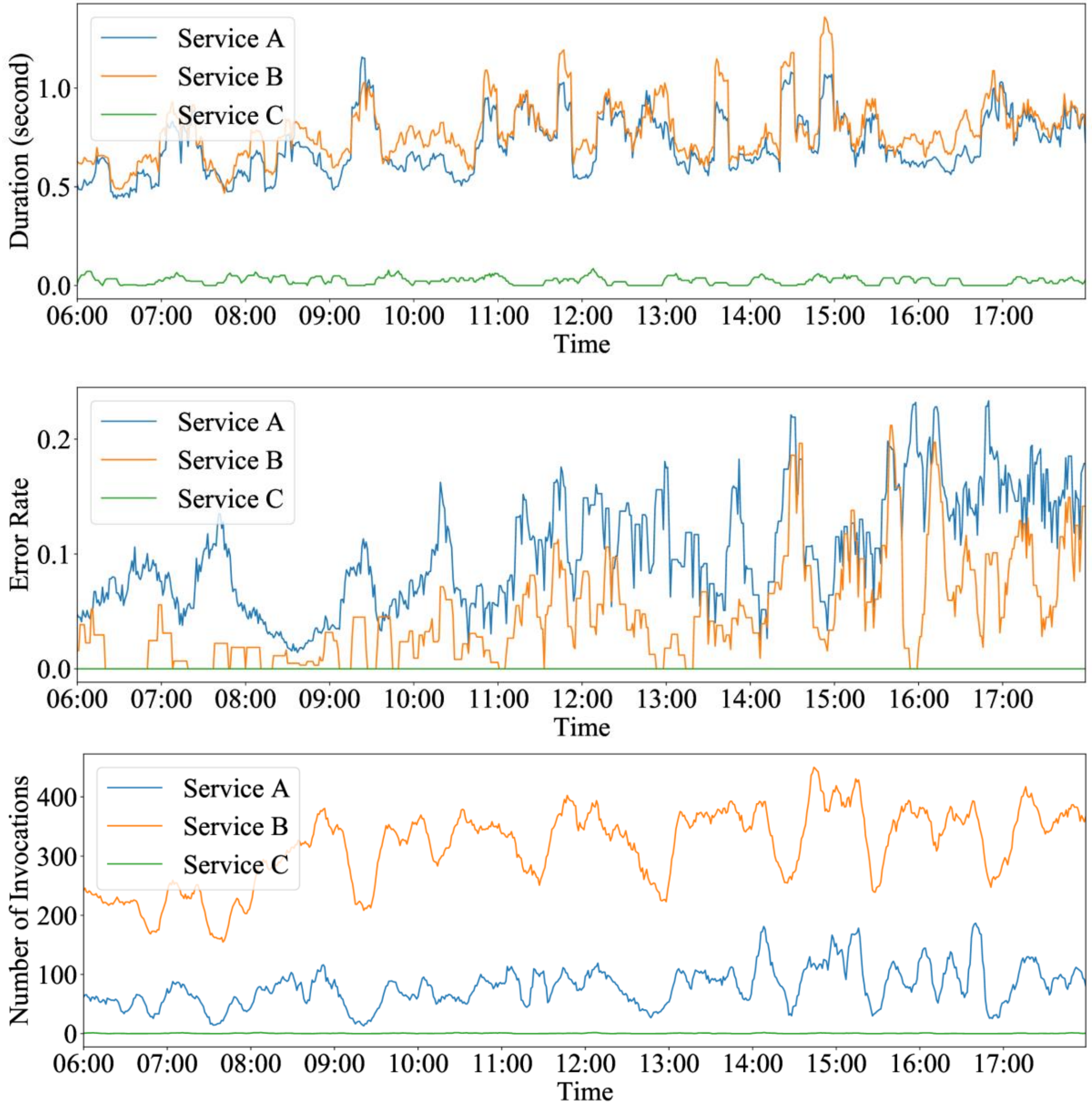
<u>Because each team only have a local view of the whole system.</u>

Current practice is inefficient and dependent on the human experience.

Intensity of Service Dependency

- The *intensity of dependency* between $A \rightarrow B$ is higher than the intensity of dependency between $A \rightarrow C$, due to
  - Functionality
  - Fault tolerance





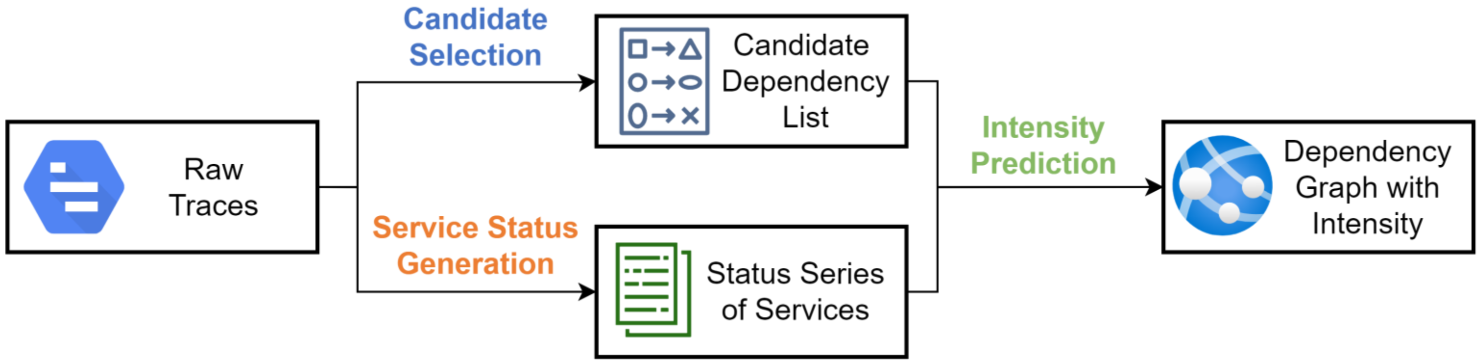
Intensity of Service Dependency

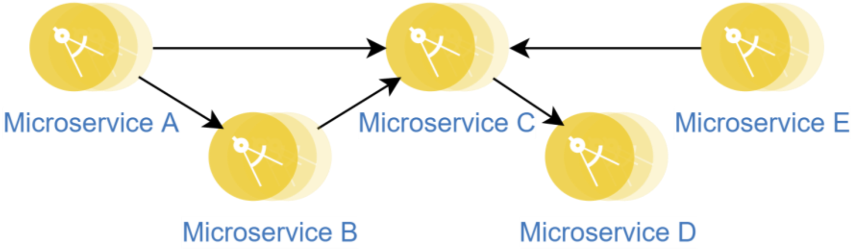
We define the intensity of dependency between two services as how much the status of the callee service influences the status of the caller service.

- Intensity is inherently determined by the program logic of services.
- Manual maintenance of intensity is hard due to the fast-evolving nature.
- But we could **predict** the intensity of dependency from traces.

Outline

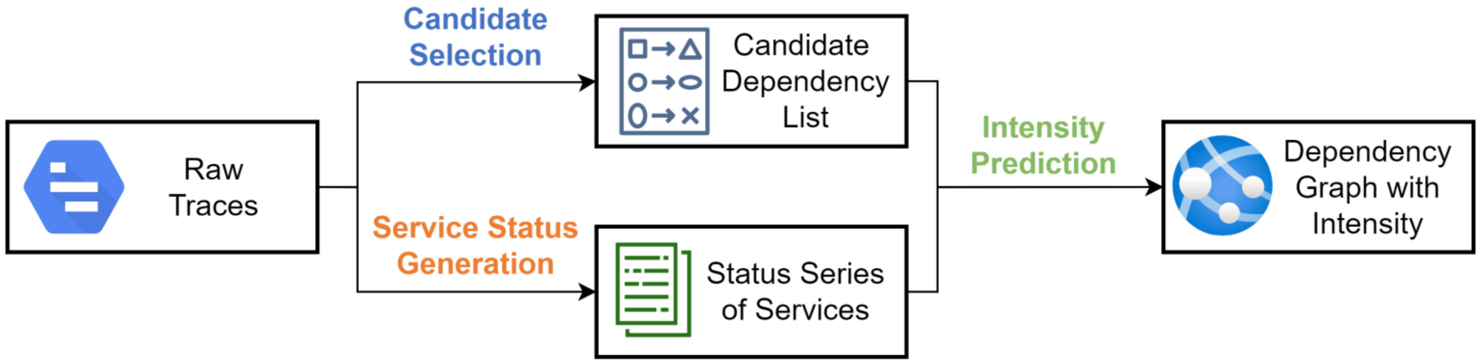
Candidate Selection

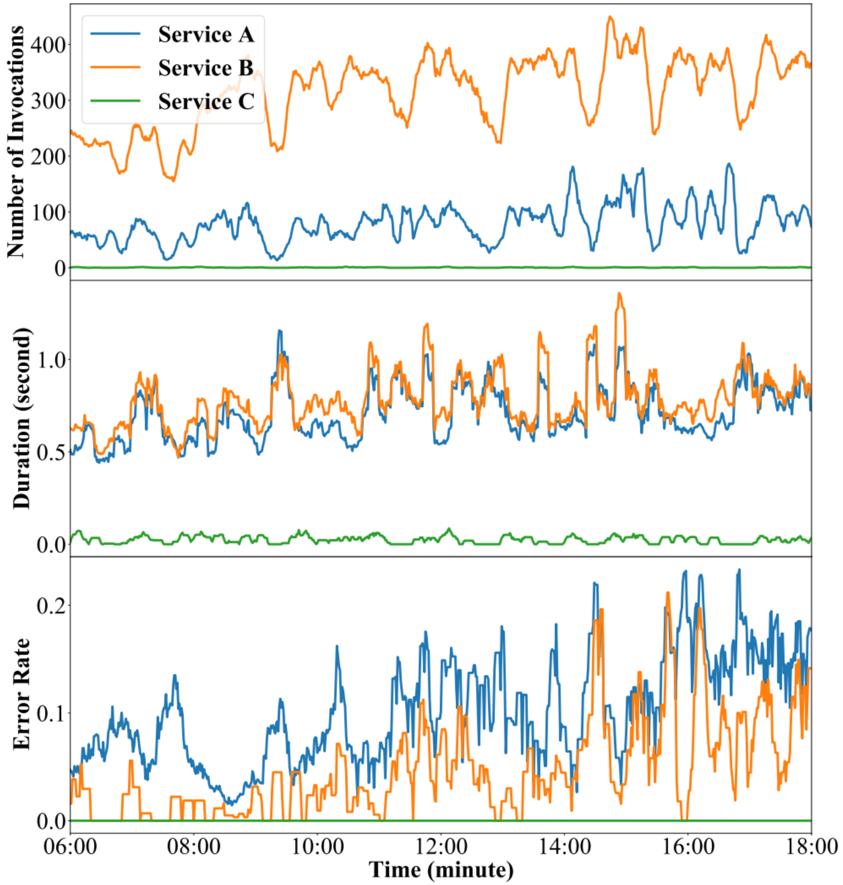




- Objective
  - Select the candidate invocation pairs ($caller$, $callee$) from raw traces where $caller$ directly invokes $callee$.
- Method
  - Iterate over all spans to get the invocation pairs.
  - Get the invocation pairs if the cloud system have a centralized database of invocation.

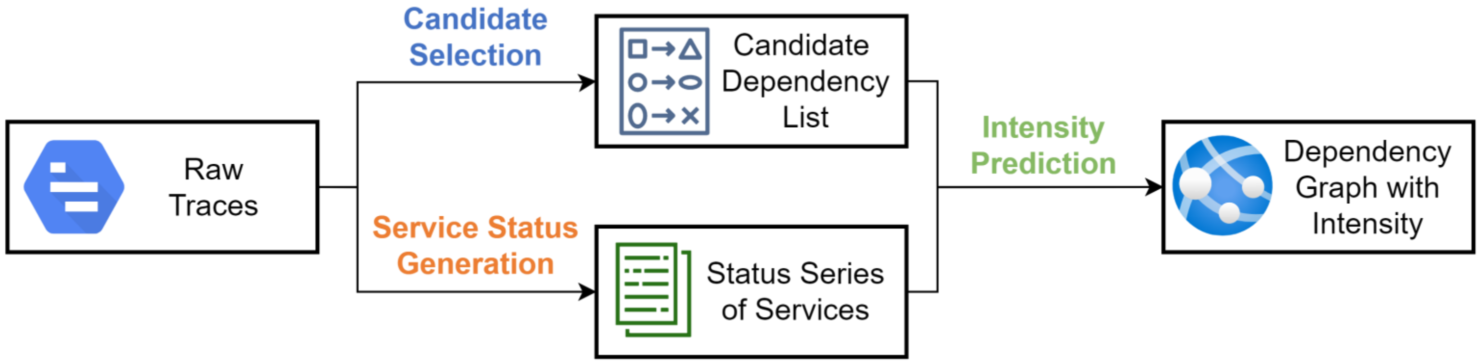
Service Status Series Generation





- Three aspects of indicators of service status
  - *Number of Invocations*
  - *Durations of Invocations*
  - *Error of Invocations*
- Method: calculate the number of invocations, average duration, and error rate of all spans of a service in a fixed time interval. (e.g., 1 minute)

Intensity Prediction



- Idea: the more similar two services' status series are, the higher the intensity is.
- Method
  - Dynamic Status Warping
  - Similarity Normalization & Aggregation

$$d_{status}^{(P_i,C_i)} = \frac{d_{status}^{(P_i,C_i)} - \min(d_{status}^{(P,C)})}{\max(d_{status}^{(P,C)}) - \min(d_{status}^{(P,C)})} \qquad I^{(P_i,C_i)} = \frac{1}{3}\sum_{status \in S} d_{status}^{(P_i,C_i)},\ S = \{invo, err, dur\}$$

Algorithm 1: Dynamic Status Warping

Input: The status series of caller service and callee service $status^P, status^C$; duration series of callee dur^C, estimated round trip time δ_{rtt}, max time drift δ_d

Output: The similarity between two status series

Set the warping window $w = \max(dur^C) + \delta_{rtt}$
$M = length(status^C)$
$N = length(status^P)$
Initialize the cost matrix $\mathbf{C} \in \mathbb{R}^{M\times N}$, set the initial values as $+\infty$
$\mathbf{C}_{1,1} = (status_1^P - status_1^C)^2$
for $i = 2 \dots \min(\delta_d, M)$ **do** // Initialize the first column
  $\mathbf{C}_{i,1} = \mathbf{C}_{i-1,1} + (status_1^P - status_i^C)^2$
end
for $j = 2 \dots \min(w + \delta_d, N)$ **do** // Initialize the first row
  $\mathbf{C}_{1,j} = \mathbf{C}_{1,j-1} + (status_j^P - status_1^C)^2$
end
for $i = 2 \dots M$ **do**
  for $j = \max(2, i - \delta_d) \dots \min(N, i + w + \delta_d)$ **do**
    $\mathbf{C}_{i,j} = \min(\mathbf{C}_{i-1,j-1}, \mathbf{C}_{i-1,j}, \mathbf{C}_{i,j-1}) + (status_j^P - status_i^C)^2$
  end
end
return $\mathbf{C}_{M,N}$

Outline

1 Background

2 Motivation

3 Methodology

4 Experiment

5 Case Study

Experiment Settings

- Dataset
  - *Industry*[1] : Production Huawei Cloud traces.
  - *TT*[2] : Simulated traces by the Train-Ticket benchmark.

TABLE II
DATASET STATISTICS.

Dataset	TT	Industry
# Microservices	25	192
# Spans	17,471,024	About 1.0e10
# Strong	18	67
# Weak	1	8

[1] We only labeled 75 dependencies that the engineers are familiar with.
[2] FudanSELab/train-ticket: Train Ticket - A Benchmark Microservice System (github.com)

RQ1: Effectiveness of Intensity Prediction

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT METHODS ON TWO DATASETS

Dataset	Method	Metric		
		CE	MAE	RMSE
TT	Pearson	0.6872	**0.3305**	0.4388
	Spearman	0.7512	0.3735	0.4697
	Kendall	0.6464	0.3749	0.4577
	AID	**0.4562**	0.3435	**0.3859**
Industry	Pearson	0.6076	0.4524	0.4563
	Spearman	0.6030	0.4501	0.4537
	Kendall	0.6258	0.4636	0.4656
	AID	**0.3270**	**0.1751**	**0.3044**

- Parameter Settings
  - Bin size $\tau = 1\ min$
  - Estimated round trip time $\delta_{rtt} = 0$
  - Max time drift
    - $\delta_d = 1\ min$ (for Industry dataset)
    - $\delta_d = 0\ min$ (for TT dataset)

RQ2 & RQ3: Ablation Study

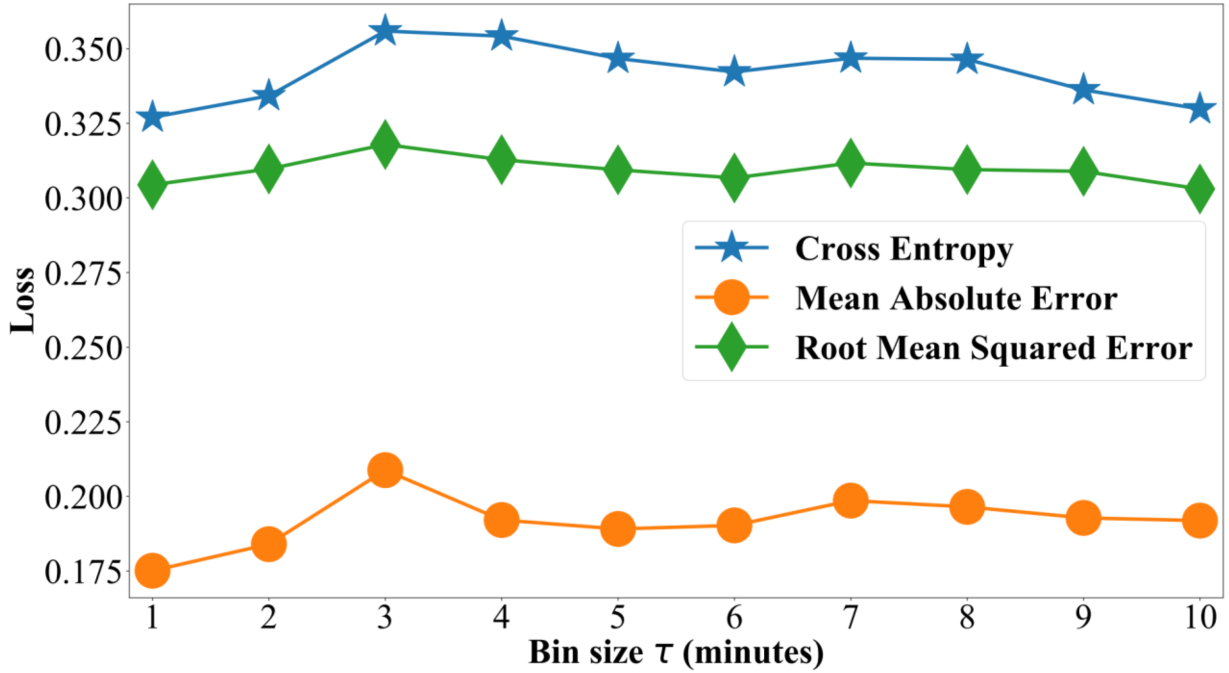


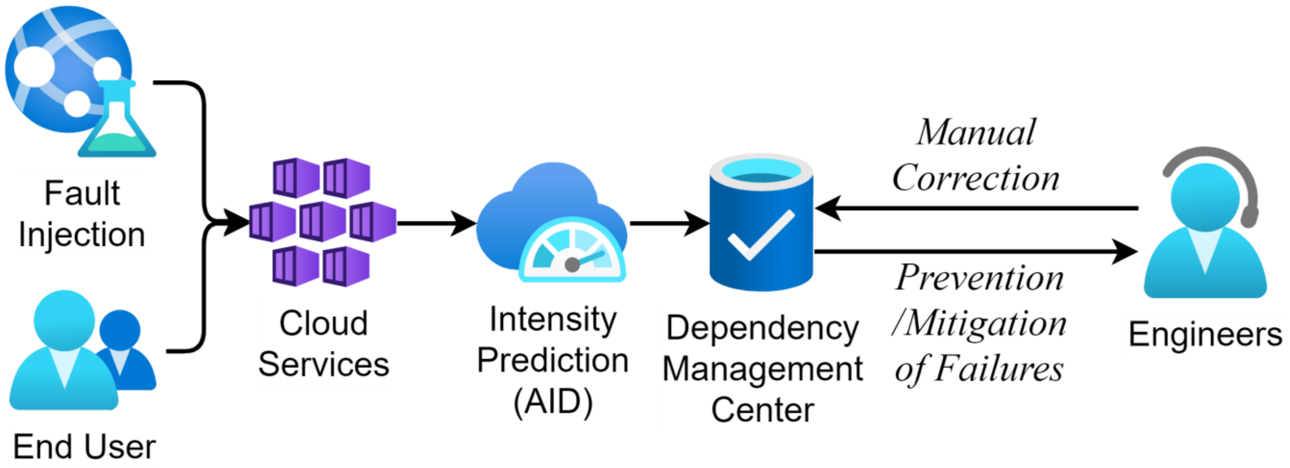
TABLE IV
THE IMPACT OF DIFFERENT SIMILARITY MEASURES

Dataset /Bin size	Method	Metric		
		CE	MAE	RMSE
TT /1min	AID_{DSW}	0.4562	0.3435	0.3859
	AID_{DTW}	0.4494	0.3467	0.3832
Industry /1min	AID_{DSW}	0.3270	0.1751	0.3044
	AID_{DTW}	0.3584	0.1996	0.3169

Outline

Use Cases of AID

- Mitigation of Cascading Failures
  - Limit the traffic to critical cloud services.
  - Recover the dependencies marked as “strong” first.
- Optimization of Dependencies
  - Dependency management system detects strong dependencies and reminds engineers.



Thank you!

ARISE
Automated Reliable Intelligent Software Engineering

香港中文大學
The Chinese University of Hong Kong