中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

TraceMesh: Scalable and Streaming Sampling for Distributed Traces

Zhuangbin Chen[1], Zhihan Jiang[2], Yuxin Su[1], Michael R. Lyu[2], Zibin Zheng[1]

[1]Sun Yat-sen University

[2]The Chinese University of Hong Kong

chenzhb36@mail.sysu.edu.cn

Outline

- **Introduction & Motivation**
- **TraceMesh Overview & Design**
- **Evaluation**
- **Conclusion**

Introduction & Motivation

Shifting to Microservices

Traditional applications

Coupled & Central

Monolithic

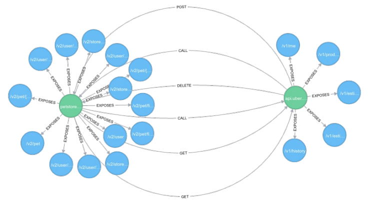
- Complex app
- Centralized deployment
- Single point of failure
- Scaling done as a whole

Shifting to

Microservice applications

Decoupled & Distributed

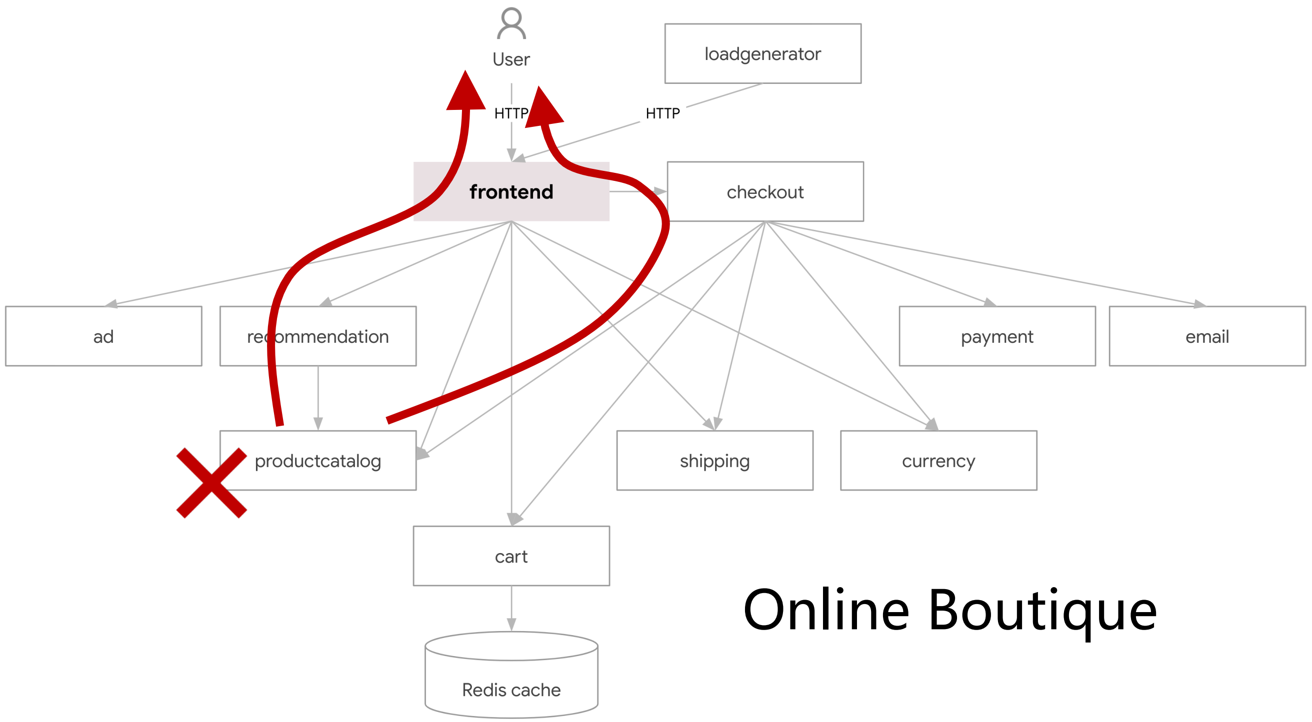
Microservices



- Multiple microservices
- Fine-grained management
- Fault isolation
- Individual scaling

Complex Reliability Management

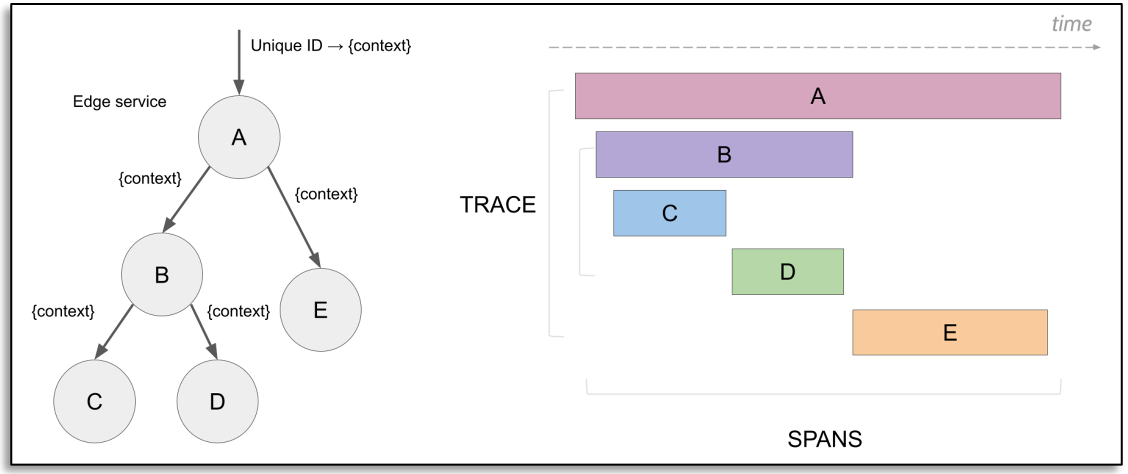
- E2E behavior can be affected by any microservice
- Failures cascading effect
- Symptoms and root causes can be far apart



Online Boutique

Distributed Tracing

- A directed tree recording executions across microservices
- Trace events: time, operations, messages, attributes
- Critical for performance monitoring, root cause analysis, etc.



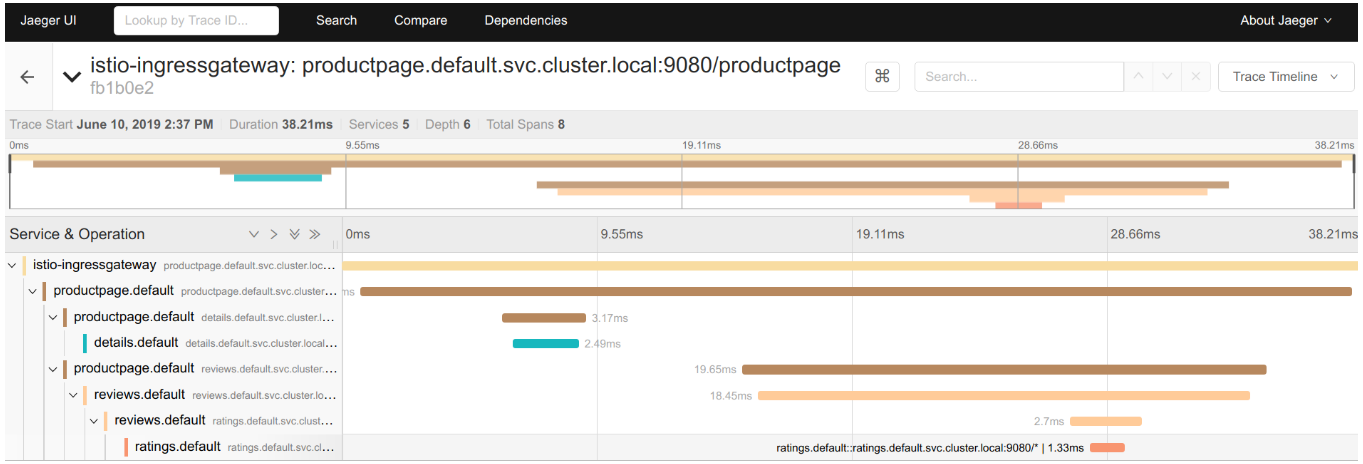
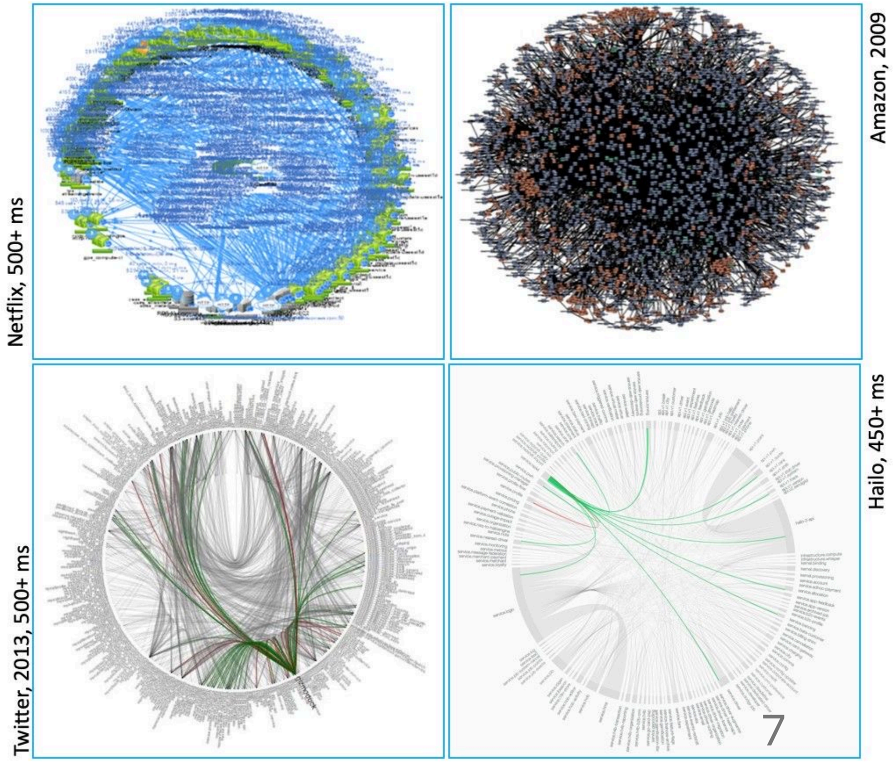


Figure 1 source: https://medium.com/nikeengineering/hit-the-ground-running-with-distributed-tracing-core-concepts-ff5ad47c7058
Figure 2 source: https://istio.io/latest/docs/tasks/observability/distributed-tracing/jaeger/

Trace Data are Large in Size

- Complex microservices systems
  - Uber is composed of thousands of microservices
  - WeChat system hosts more than 3,000 services

- Huge volume of trace data
  - E.g., Google generates about 1,000 TB of raw traces on a daily basis
  - **Significant overhead to trace generation, collection, and ingestion**



Most Trace Data are Useless

- Most traces are recurring and useless
  - Cloud services often guarantee more than 99% uptime in their SLAs
  - Only edge-case traces are interesting and **should be sampled**
  - **By definition, these traces are rare**

AMAZON ELASTIC COMPUTE CLOUD (EC2) (INSTANCE LEVEL)

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.5% but equal to or greater than 99.0%	10%
Less than 99.0% but equal to or greater than 95.0%	30%
Less than 95.0%	100%

Amazon Compute Full SLA

Compute | Containers

https://aws.amazon.com/legal/service-level-agreements/

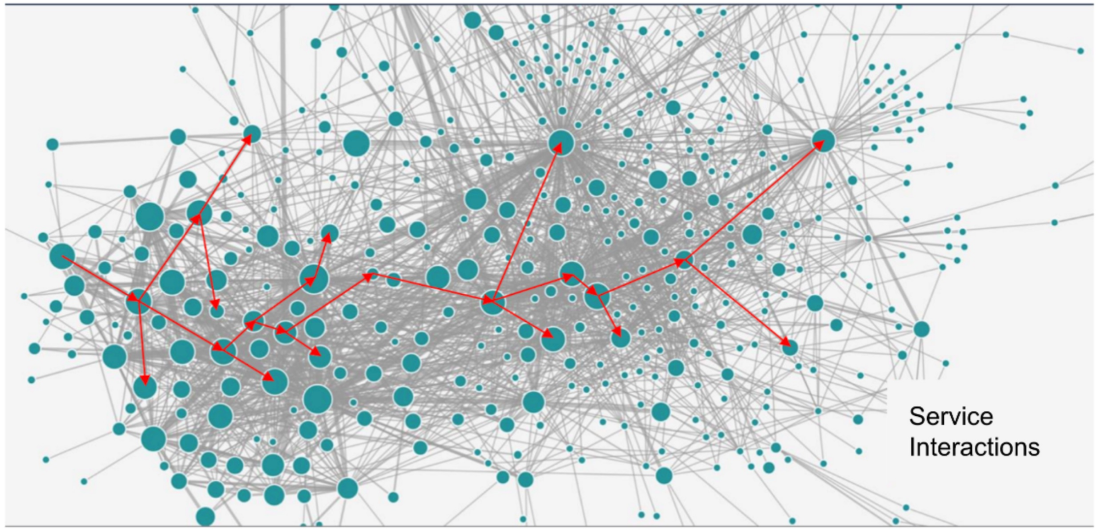
Trace Sampling

- Head-based sampling
  - Decision made **prior** to request execution
  - Uniformly sample traces at random with a sampling rate, e.g., 1%
  - Could result in common-case traces

- Tail-based sampling
  - Decision made **after** request execution
  - Biased sampling based on latency, HTTP status code, etc.
  - Learning-based approaches to identify uncommon traces

Challenges of Tail-based Trace Sampling

- Large feature dimension
  - Curse of dimensionality
  - Compromised performance and scalability



Microservices call graph collected at Uber

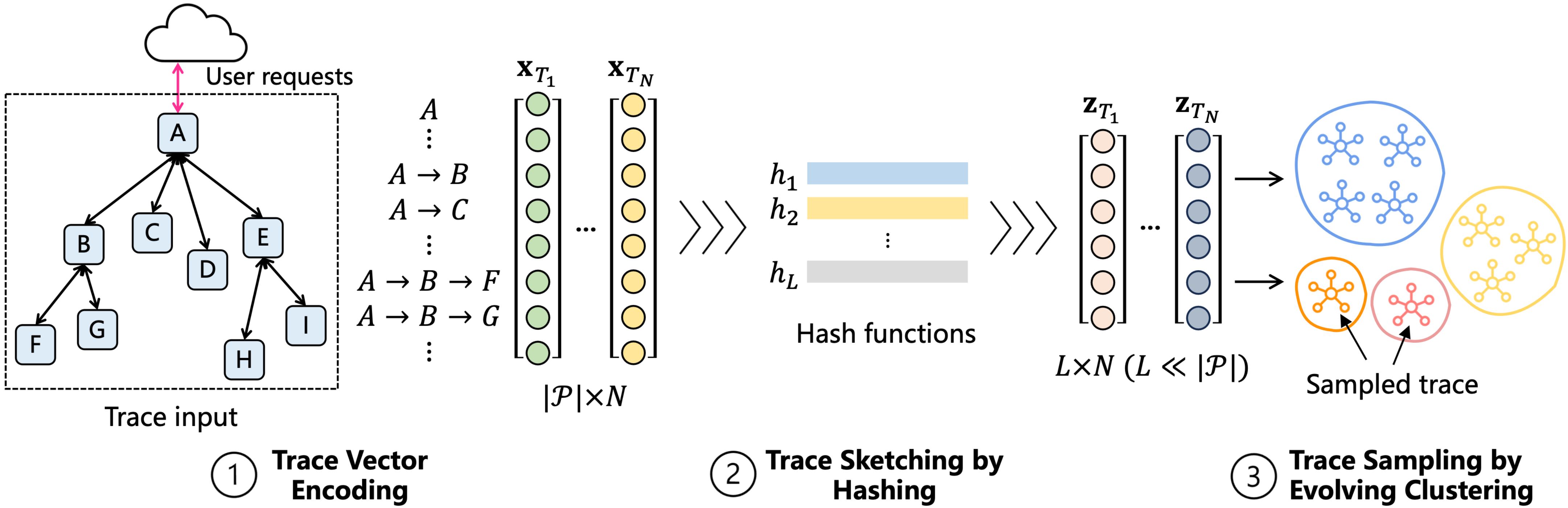
- Frequent pattern change
  - Microservices undergo continuous updates
  - New trace patterns could emerge

Not well addressed in existing approaches!

Figure source: *CRISP: Critical Path Analysis of Large-Scale Microservice Architectures*. Zhang et al. USENIX ATC 2022.

TraceMesh Overview & Design

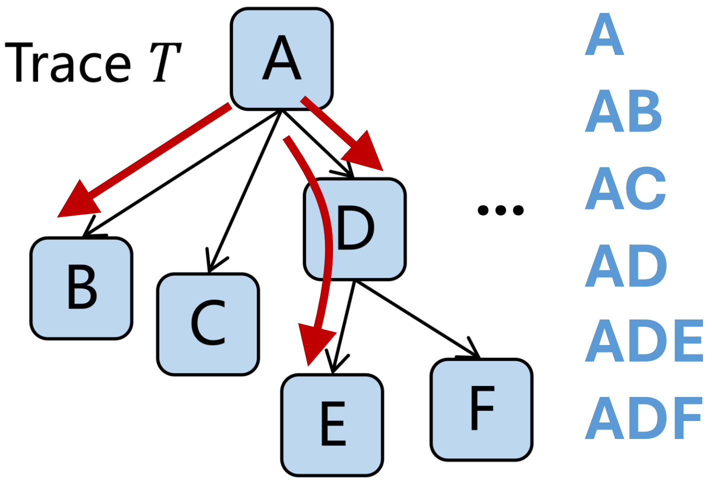
TraceMesh Overview

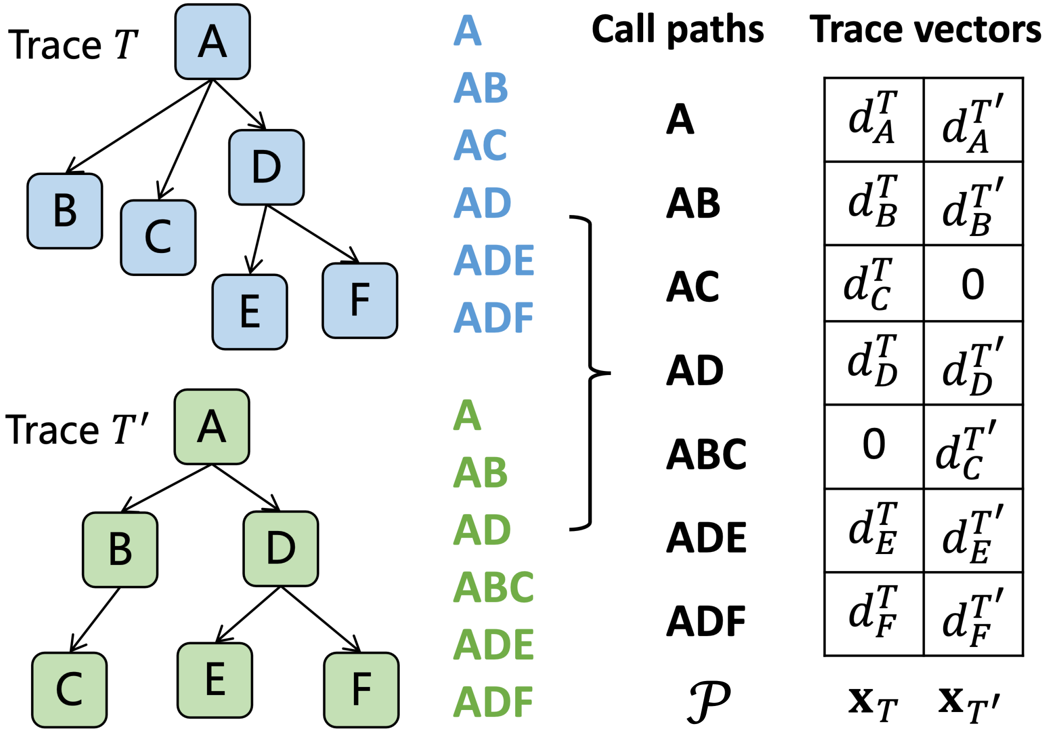


1. Transform graph traces into numerical feature vectors
2. Perform streaming trace vector encoding
3. Adaptively sample target traces by clustering

Step 1: Trace Vector Encoding

- Two types of trace features are extracted:
  - Structural feature
  - Temporal feature

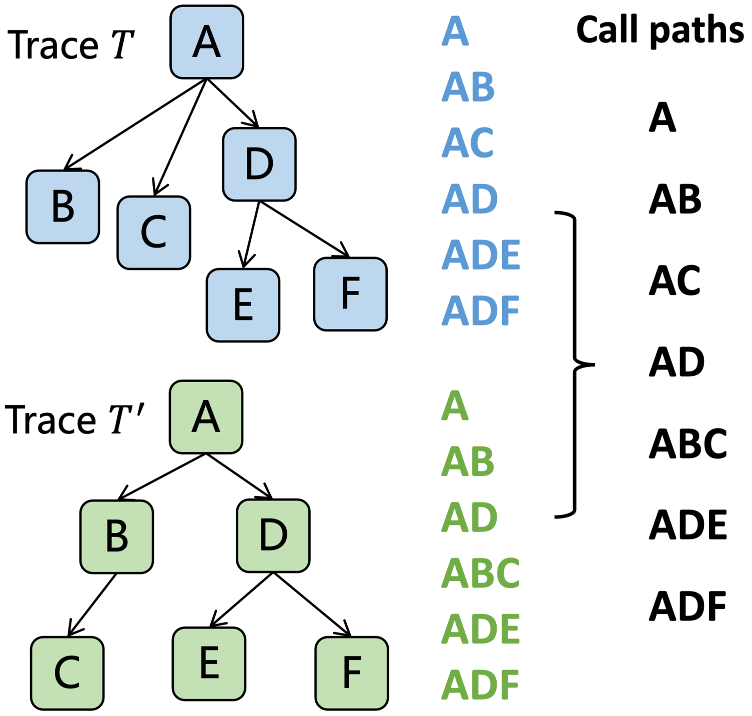




The complete set of call paths can be large and dynamic!

Step 2: Trace Sketching by Hashing

- The first challenge:



$\mathcal{P}$ can be very large

Locality-Sensitive Hashing (LSH)

Project high-dimensional vectors into a low-dimensional space while preserving their similarity

Step 2: Trace Sketching by Hashing

- The projection process for a trace vector $\mathbf{x}$ in $\mathbb{R}^{|\mathcal{P}|}$:
  1. Instantiate L projection vectors $\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_L \in \{+1, -1\}^{|\mathcal{P}|}$
  2. Compute trace sketch vector as $\mathbf{z} = \left[h_{\mathbf{r_1}}(\mathbf{x}), \dots, h_{\mathbf{r}_L}(\mathbf{x})\right] \in \{+1, -1\}^L$:

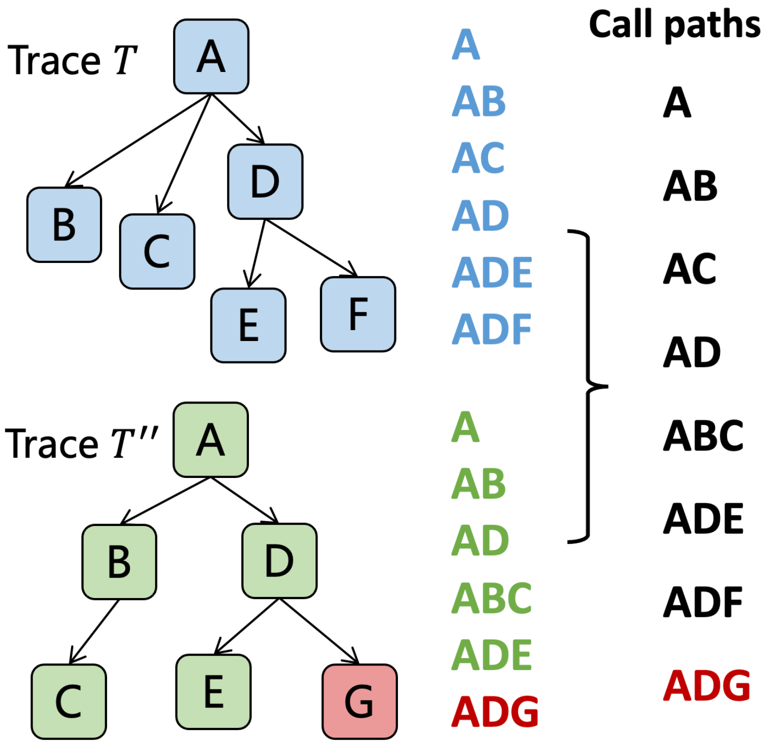
$$h_{\mathbf{r}_l}(x) = \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{r}_l \geq 0 \\ +1 & \text{if } \mathbf{x} \cdot \mathbf{r}_l < 0 \end{cases} \quad \text{i.e., } h_{\mathbf{r}_l}(x) = \text{sign}(\mathbf{x} \cdot \mathbf{r}_l)$$

- Calculate the similarity for two traces by quantifying the agreement level between the sketch vectors:

$$sim(T, T') \propto \frac{|\{l : \mathbf{z}_T(l) = \mathbf{z}_{T'}(l)\}|}{L}$$

Step 2: Trace Sketching by Hashing

- The second challenge:



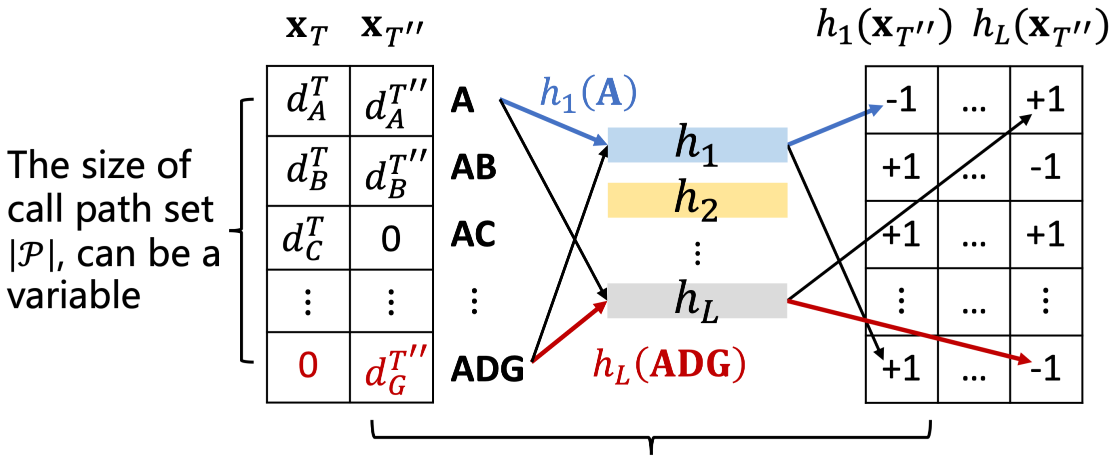
$\mathcal{P}$ can be dynamic

StreamHash

Replace project vectors with hash functions to deterministically map any call paths to $\{+1, -1\}$

Streaming Trace Vector Encoding

- L hash functions $h_1, h_2, \dots, h_L$ uniformly picked from a family $\mathcal{H}$ of hash functions



$$\mathbf{z}(l) = \operatorname{sign}\big(\mathbf{y}(l)\big) = \operatorname{sign}(\mathbf{x} \cdot h_l), l = 1, \dots, L$$

Fixed trace vector size L

$\mathbf{z}_{T''}(1)$	$\operatorname{sign}(-1 * d_A^{T''} + \cdots + (+1) * d_G^{T''})$
⋮	⋮
$\mathbf{z}_{T''}(L)$	$\operatorname{sign}(+1 * d_A^{T''} + \cdots + (-1) * d_G^{T''})$

The trace T'' with a new path ADG can be seamlessly encoded as $\mathbf{z}_{T''}$

Step 3: Trace Sampling by Evolving Clustering

- A **density-based clustering** approach for stream data

- Each **micro-cluster (MC)** has three attributes:
  - A weight w, determined by the traces inside it and time
  - A center c, the weighted center of the traces inside it
  - A radius r, the weighted avg distance from points to c

- Two types of MCs with different weight constraints:
  - **Potential MC (PMC)**: contains frequent and usual traces
  - **Outlier MC (OMC)**: contains potentially outlier traces

Step 3: Trace Sampling by Evolving Clustering

- When a new trace T arrives, TraceMesh
  - Merges it into the nearest PMC, and samples it with probability:

$$p_{\mathrm{T}} = \mathcal{B} \times (1 - \frac{w_p^*}{\sum_{i=0}^{N_{pmc}} w^{(i)}})$$

$\mathcal{B}$: sampling budget

  - If failed, merges it into the nearest OMC, and samples it
  - If failed, makes T a new OMC, and samples it

In this process, the **weight and role** of each MC will change. TraceMesh **dynamically adjusts the sampling strategy** to identify the target traces without over-sampling recurring traces.

Evaluation

Datasets

- Traces from open-source microservices
  - Train Ticket
  - Online Boutique
- Traces from production systems

DATASET STATISTICS

Dataset	#Span	#Train	#Test	#Label
Train Ticket	201	653	31,814	624
Online Boutique	43	1,024	78,931	225
Industry	1,308	11,847	497,439	2,453

Train Ticket: https://github.com/FudanSELab/train-ticket
Online Boutique: https://github.com/GoogleCloudPlatform/microservices-demo

Metrics & Baselines

- Metrics

$$\textbf{Coverage} = \frac{\#\ target\ traces\ sampled}{\#\ target\ traces} \qquad \textbf{Sampling rate} = \frac{\#\ traces\ sampled}{\#\ traces}$$

- Baselines
  - Uniform sampling
  - Perch [SoCC 2018]
  - Sifter [SoCC 2019]
  - Sieve [ICWS 2021]
  - SampleHST [NOMS 2023]

Experimental Results

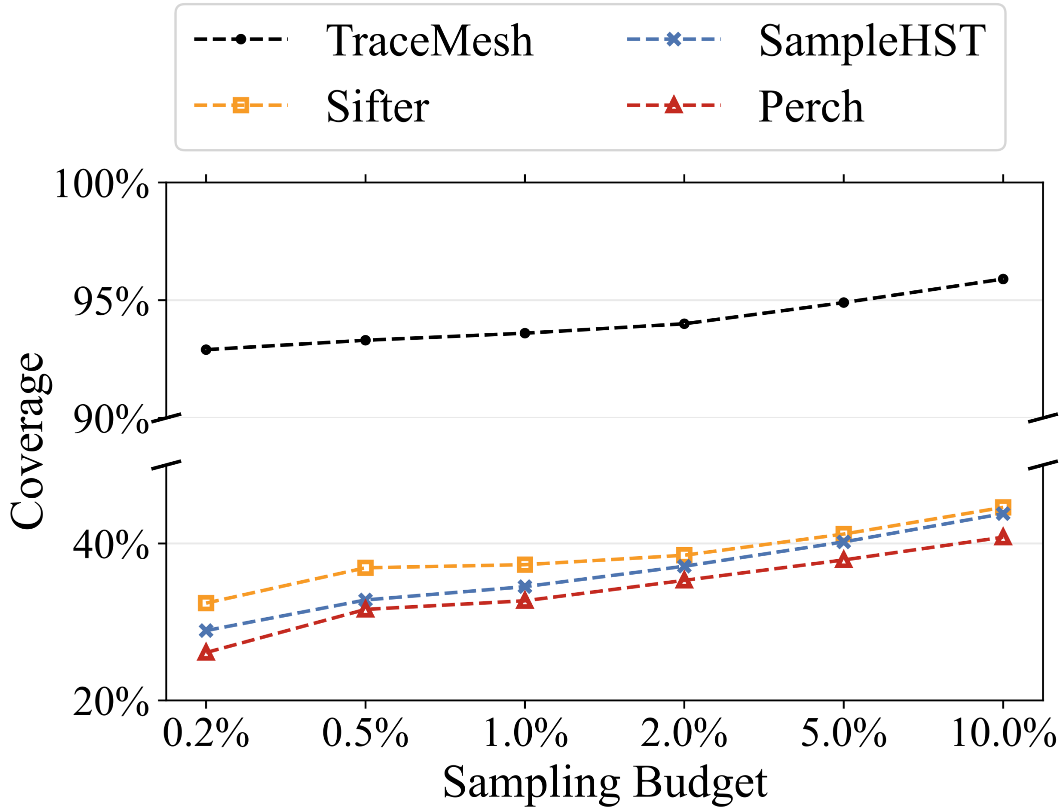
- TraceMesh achieves the best coverage
- TraceMesh has a relatively good sampling rate

EXPERIMENTAL RESULTS OF TRACE SAMPLING

Method	Train Ticket Coverage	Train Ticket Sampling Rate	Online Boutique Coverage	Online Boutique Sampling Rate	Industry Coverage	Industry Sampling Rate
Uniform	1.1%	**1.00%**	1.1%	1.00%	1.0%	1.00%
Sieve	71.8%	6.24%	82.2%	3.91%	61.1%	3.29%
Sifter	50.6%	1.28%	42.2%	1.13%	37.3%	1.16%
SampleHST	46.2%	1.19%	38.2%	1.07%	34.5%	1.10%
Perch	42.9%	**1.00%**	35.1%	**0.98%**	32.7%	0.95%
TRACEMESH	**98.7%**	2.34%	**100%**	1.35%	**93.6%**	**0.83%**

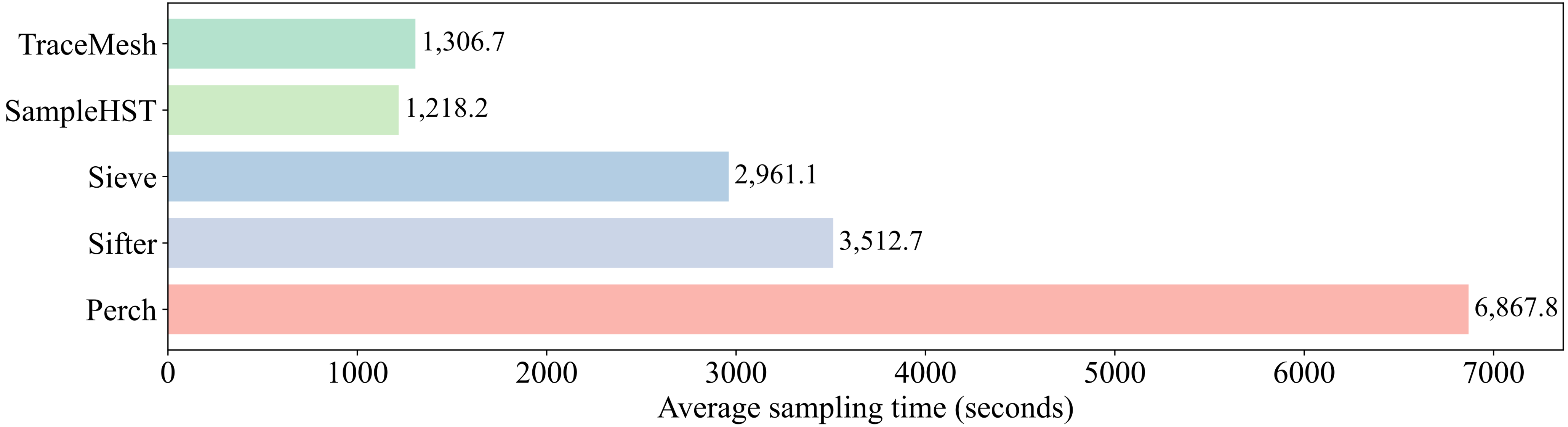
Coverage vs. Sampling Budget

- Coverage increases with larger sampling budget
- TraceMesh performs trace sampling more accurately



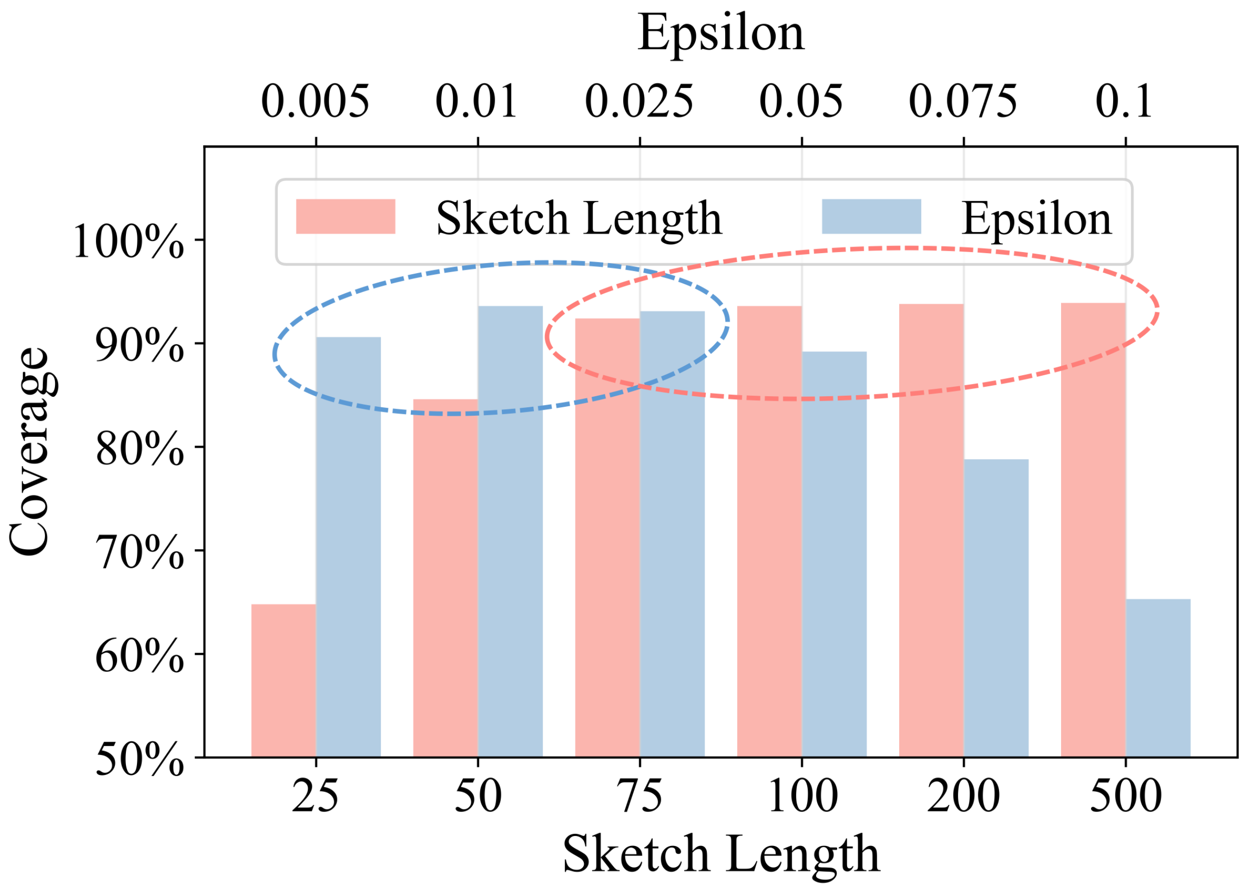
Efficiency

- TraceMesh performs trace sampling efficiently



Parameter Sensitivity

- TraceMesh can achieve stable performance



Conclusion

Conclusion

- TraceMesh, a **scalable** and **streaming** trace sampler
  - Large feature dimension: LSH for dimension reduction
  - Frequent pattern changes: StreamHash and evolving clustering

- TraceMesh outperforms baselines in both sampling accuracy and efficiency

TraceMesh: Scalable and Streaming Sampling for Distributed Traces

Thanks!
Q & A