应用感知技术

沈俊贤
网络科学与网络空间研究院

2024.11.19

目录

- **应用感知介绍**
- 应用感知目标、难点与技术
- 应用感知插桩与BPF技术
- 应用感知技术相关研究

*There are **known knowns**; there are things we know we know.*
*We also know there are **known unknowns**; that is to say we know there are some things we do not know.*
*But there are also **unknown unknowns**— there are things we do not know we don't know.*

— U.S. Secretary of Defense Donald Rumsfeld, February 12, 2002

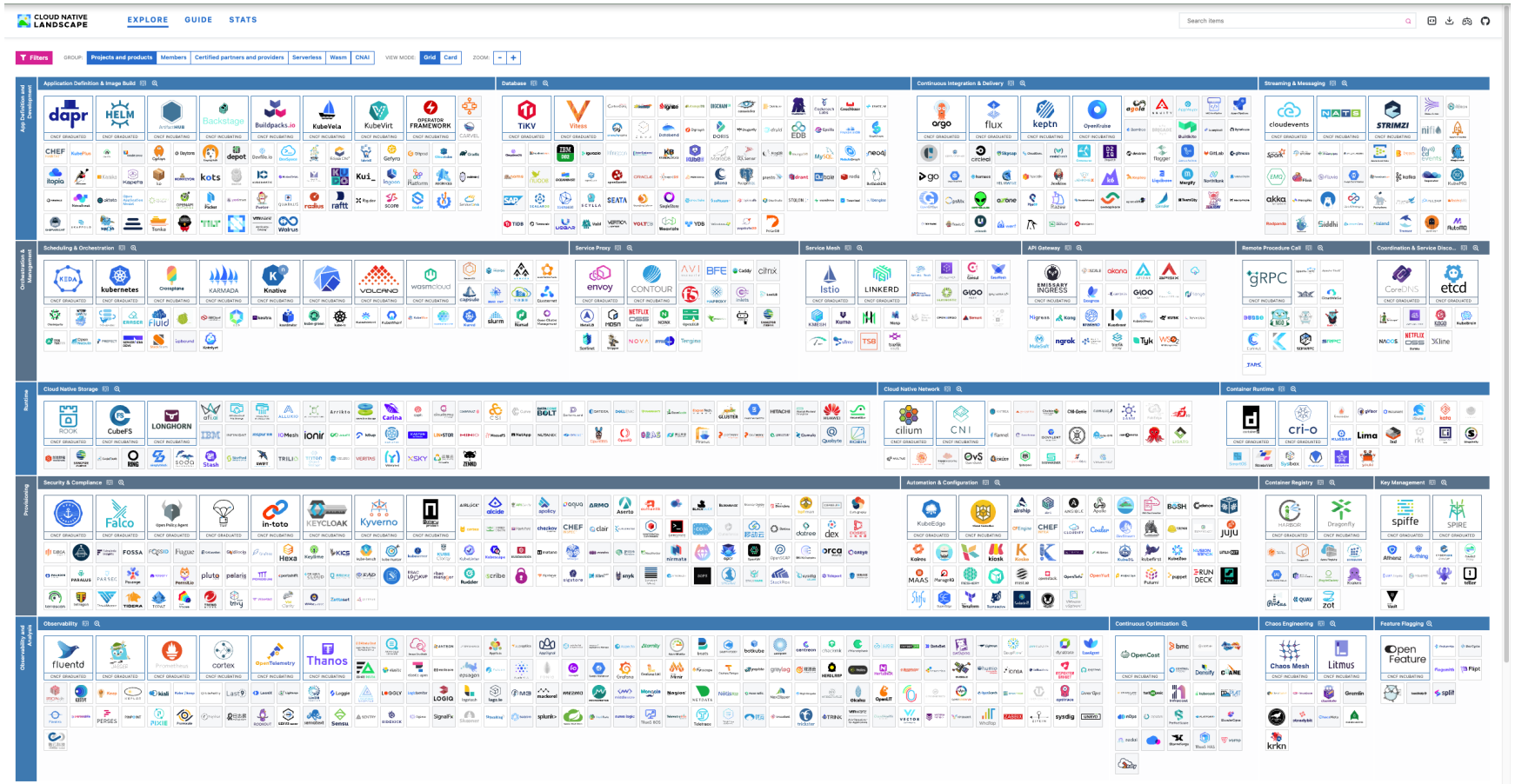
Modern Applications

- ☐ *Internet of Things*
- ☐ *Cloud Computing & Edge Computing*
- ☐ *High-Performance Computing*
- ☐ *AI & Machine Learning*
- ☐ *AR & VR*
- ☐ *Mobile Applications*
- ☐ ...

现代应用的功能、规模与复杂性不断提升！

现代应用正在重构这个世界

☐ *Cloud Native Computing Landscape*



我们需要知道应用“未知的未知”

□ 什么是应用感知？

✓应用感知是指对软件应用程序及其运行环境的状态和行为进行实时监控、分析和理解

□ 为什么要进行应用感知？

✓ 故障修复、性能优化

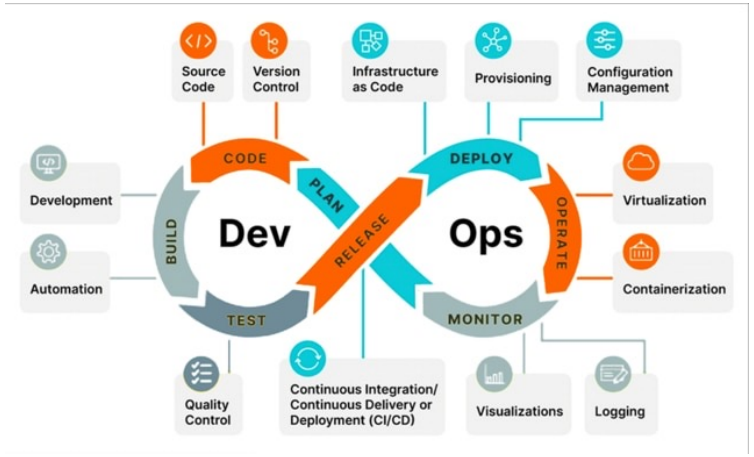
✓ 调度决策、安全防护

✓ 用户行为分析、 开发生命周期管理

AWS 事件后摘要

AWS Health Dashboard 提供全球 AWS 服务的最新运行状况信息，以及过去 12 个月的服务历史记录。当发生了对客户产生广泛而重大的影响的问题，导致很大比例的控制面板 API 调用失败，影响到很大比例的服务基础设施、资源或 API，或者是由完全停电或严重的网络故障造成的问题，AWS 承诺在问题结束后提供公开的事件后摘要 (PES)。事件后摘要将至少保留 5 年，并提供问题的影响范围、导致问题的因素以及为应对已发现的风险而采取的行动。

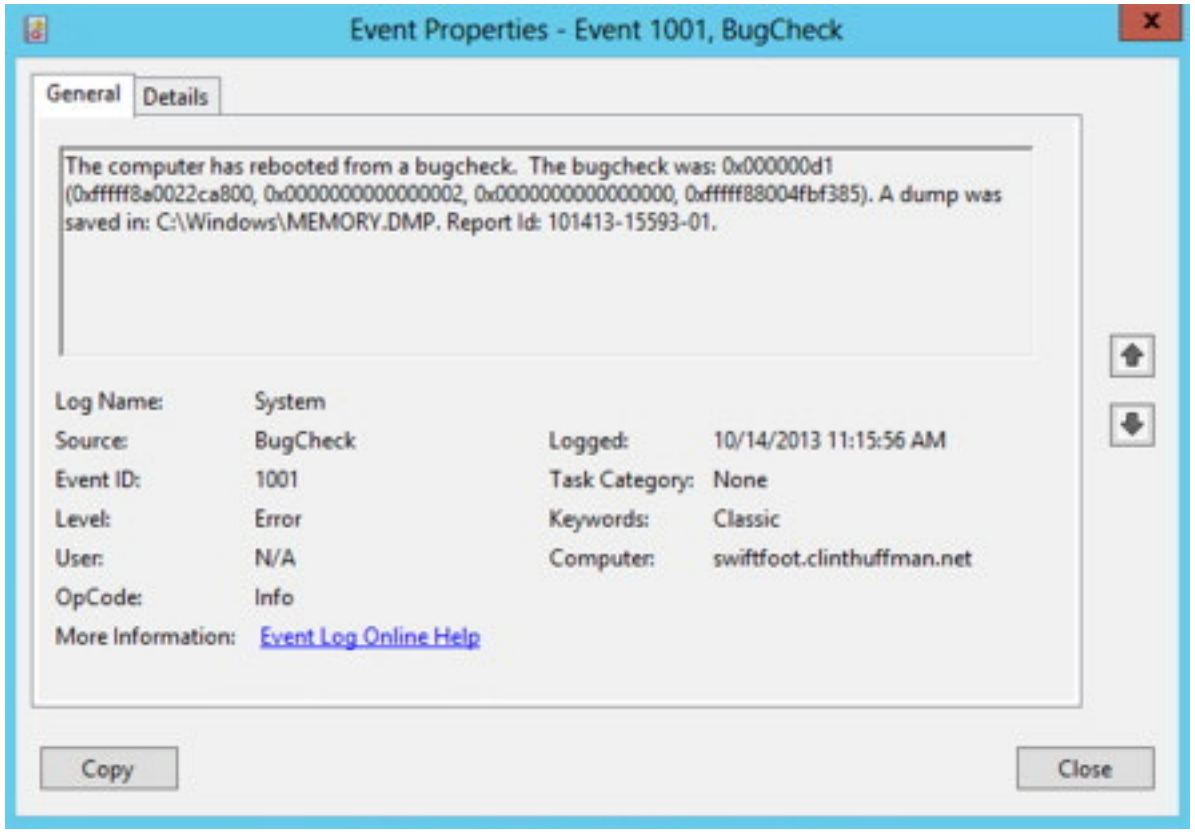
- 弗吉尼亚州北部（US-EAST-1）区域的 Amazon Kinesis Data Streams 服务事件摘要，2024 年 7 月 30 日
- 弗吉尼亚州北部（US-EAST-1）区域的 AWS Lambda 服务事件摘要，2023 年 6 月 13 日
- 弗吉尼亚州北部 (US-EAST-1) 区域的 AWS 服务事件摘要，2021 年 12 月 7 日
- 东京 (AP-NORTHEAST-1) 区域的 AWS Direct Connect 事件摘要，2021 年 9 月 2 日
- 弗吉尼亚州北部 (US-EAST-1) 区域的 Amazon Kinesis 事件摘要，2020 年 11 月 25 日
- 东京 (AP-NORTHEAST-1) 区域的 Amazon EC2 和 Amazon EBS 服务事件摘要，2019 年 8 月 23 日
- 亚太地区（首尔）区域 (AP-NORTHEAST-2) 的 Amazon EC2 DNS 解析问题摘要，2018 年 11 月 22 日。
- 弗吉尼亚州北部 (US-EAST-1) 区域的 Amazon S3 服务中断摘要，2017 年 2 月 28 日。
- 悉尼区域的 AWS 服务事件摘要，2016 年 6 月 4 日。
- 美国东部区域的 Amazon DynamoDB 服务中断及相关影响摘要，2015 年 9 月 20 日。
- 欧洲西部区域的 Amazon EC2、Amazon EBS 和 Amazon RDS 服务事件摘要，2014 年 8 月 7 日。
- Amazon SimpleDB 服务中断摘要，2014 年 6 月 13 日。
- 南美洲区域 (SA-EAST-1) 的 12 月 17 日事件摘要，2013 年 12 月 17 日。
- 美国东部区域的 2012 年 12 月 24 日 Amazon ELB 服务事件摘要，2012 年 12 月 24 日。
- 美国东部区域的 2012 年 10 月 22 日 AWS 服务事件摘要，2012 年 10 月 22 日。
- 美国东部区域的 AWS 服务事件摘要，2012 年 6 月 29 日。
- 美国东部区域的 Amazon EC2 和 Amazon RDS 服务中断摘要，2011 年 4 月 21 日



应用感知支撑了现代应用开发-部署-运行全生命周期的顺利进行

应用感知技术的发展历史

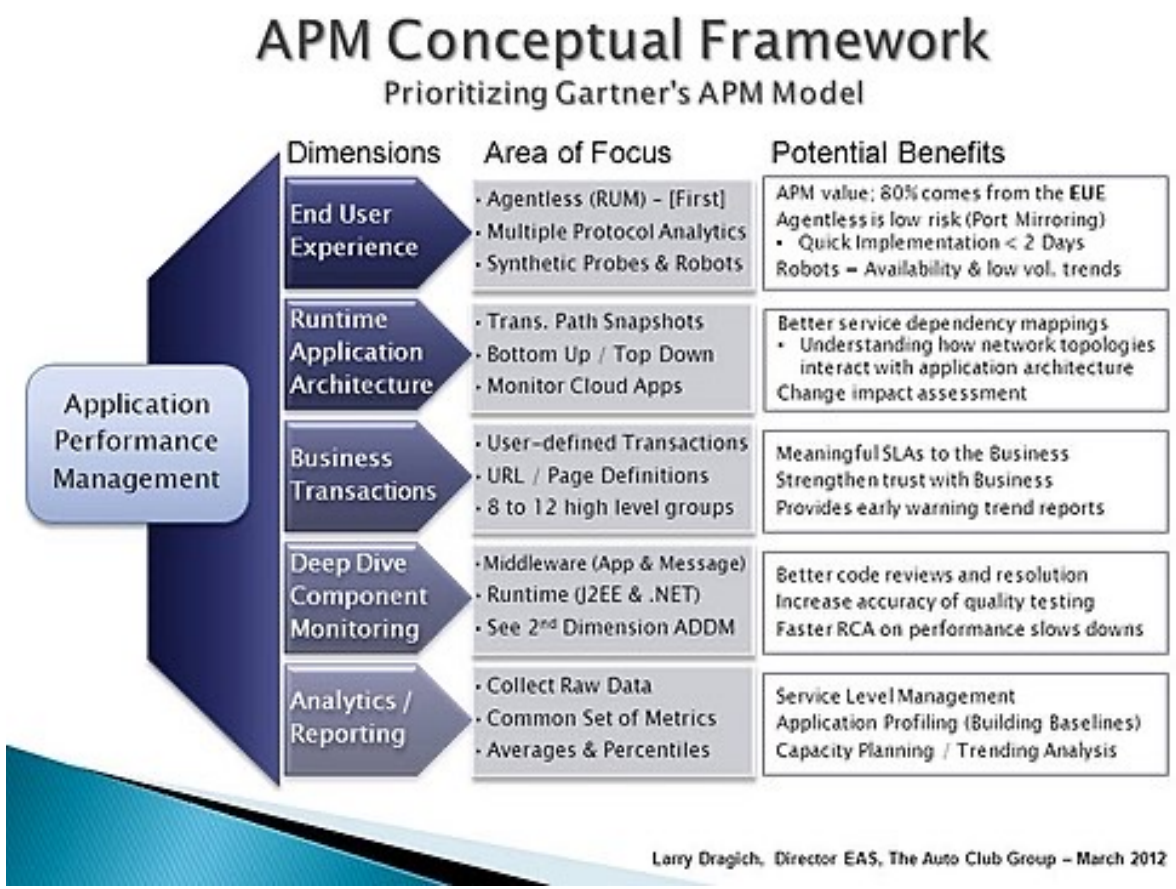
- 早期软件系统
  - ✓ 转储文件



应用感知技术的整体发展历史

- 早期软件系统
  - ✓ 转储文件

- 应用程序性能管理工具
  - ✓ *Application Performance Management (APM)*
    - ◆ 早期：硬件设备和网络通信
    - ◆ 中期：基础设施与中间件
    - ◆ 现在：应用业务本身

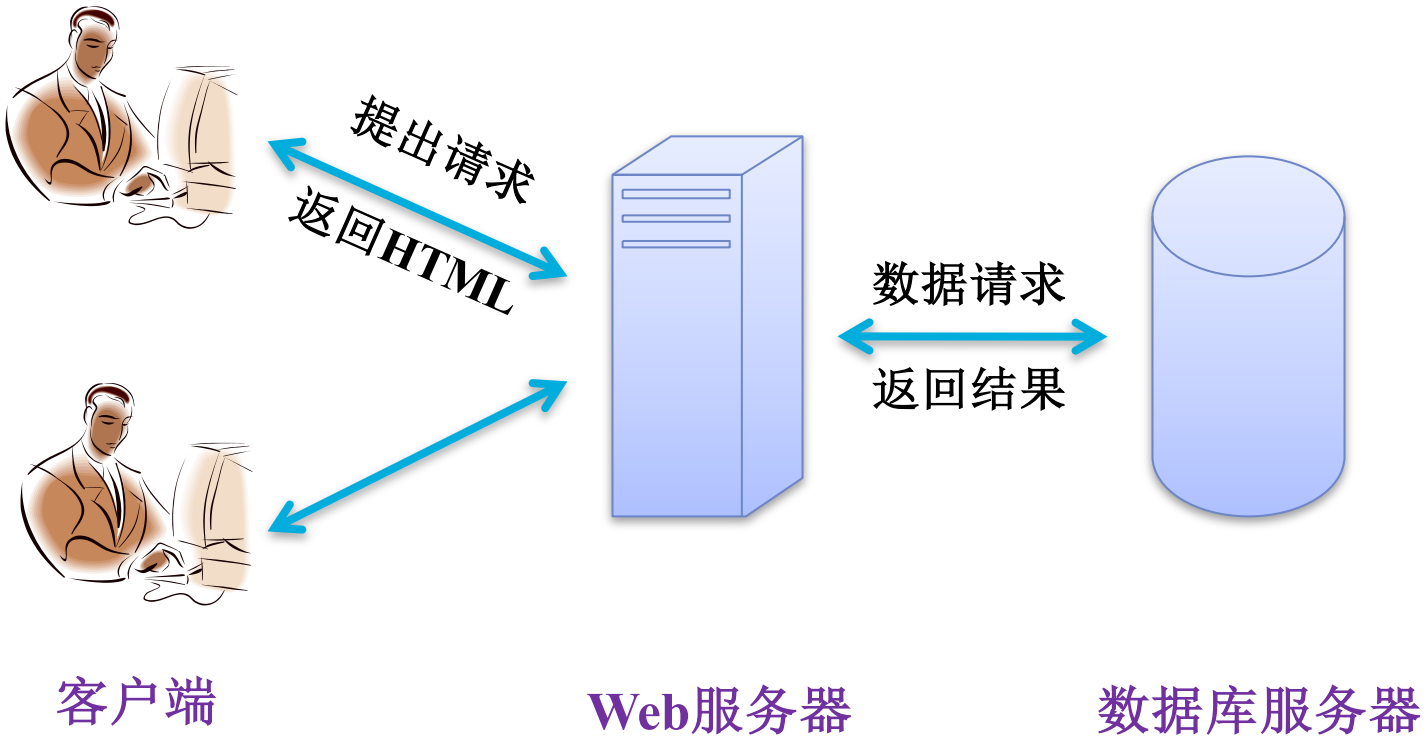


例子：铁路网上订票系统

- 为什么网络订票系统登录难、浏览慢、下单难？
  - 系统处理能力不足？
  - 什么地方存在设计瓶颈？
  - 网络带宽不足？

例子：铁路网上订票系统

□ 网络订票系统体系结构



如何了解负载特征，分析应用性能瓶颈？

目录

- 应用感知技术介绍
- **应用感知目标、难点与技术**
- 应用感知插桩与BPF技术
- 应用感知技术相关研究

We were able to speak more productively when it was clear what we were talking about.

— Peter Bourgon, February 21, 2017

应用感知的 (理想) 宏观目标

- 理解应用程序的内部工作原理
- 理解应用程序可能进入的任何系统状态
  - ✓ 甚至是那些你从未见过且无法预测的状态
- 仅通过使用和外部工具进行观测和调查
- 不需要发布任何新的定制代码来处理它
  - ✓ 因为这意味着你需要事先知道相关问题才能解释

应用感知的难点

- 主观性
- 复杂性
- 多问题并存

应用感知难点一：主观性

- 目标不同，感知的结果不同
- 例子1：飞机的性能目标
  - ✓ 哪个性能更好？
    - ◆ 单个乘客的旅行时间？
    - ◆ 1天内可以运行多少乘客？
- 例子2: 应用性能比较
  - ✓ 绝对比较法
    - ◆ *Perf(X)=(20+10)/2 = 15; Perf(Y)=(10+20)/2 = 15*
  - ✓ 相对比较法，以X为基准
    - ◆ *Perf(X)=1; Perf(Y)=(0.5+2)/2 = 1.25*
  - ✓ 相对比较法，以Y为基准
    - ◆ *Perf(Y)=1; Perf(X)=(2+0.5)/2 = 1.25*
  - ✓ 同样的数据，我们得出了三个不同的结论

飞机	华盛顿—巴黎	速度 kmph	载客量	吞吐率 kmph*person
波音747	6.5小时	610	470	286,700
协和	3小时	1350	132	178,200

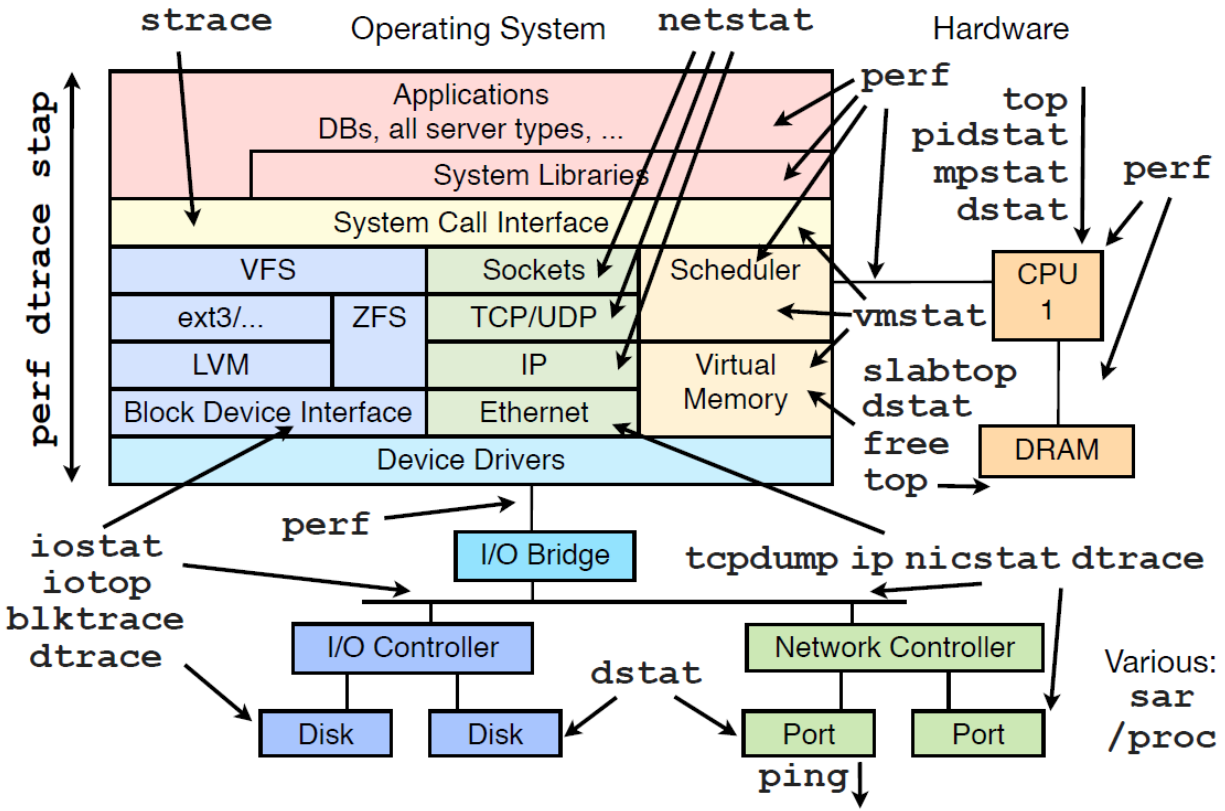
	X应用 tps	Y应用 tps
负载A	20	10
负载B	10	20

应用感知难点二：复杂性

◻ 责任范围复杂

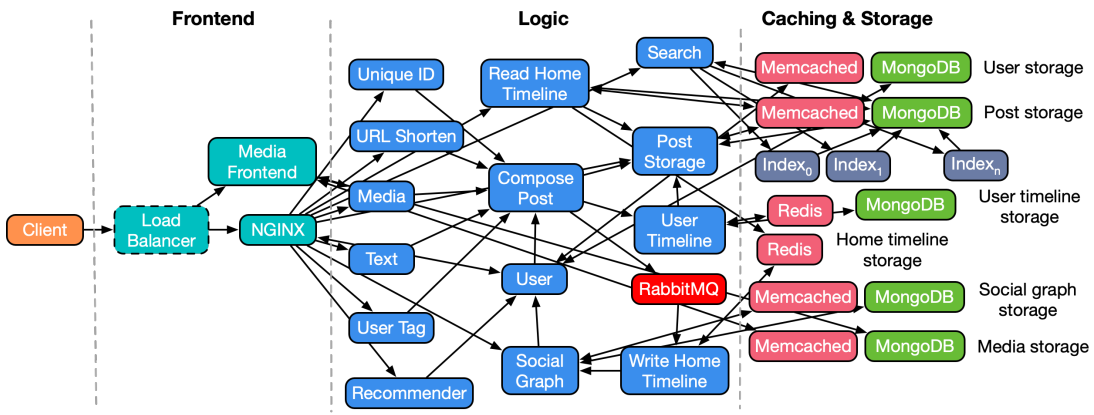
- ✓系统管理员
- ✓网站可靠性工程师
- ✓应用开发者
- ✓网络工程师
- ✓数据库管理员
- ✓网站管理员
- ✓其他支持人员
- ✓性能工程师

◻ 感知起点复杂

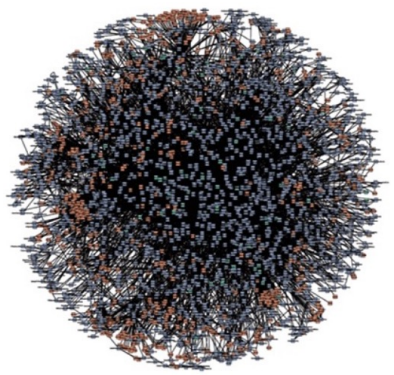


应用感知难点二：复杂性

- 应用系统复杂
  - ✓ 理清组件间的关系
  - ✓ 了解组件如何协同工作



- 生产环境复杂



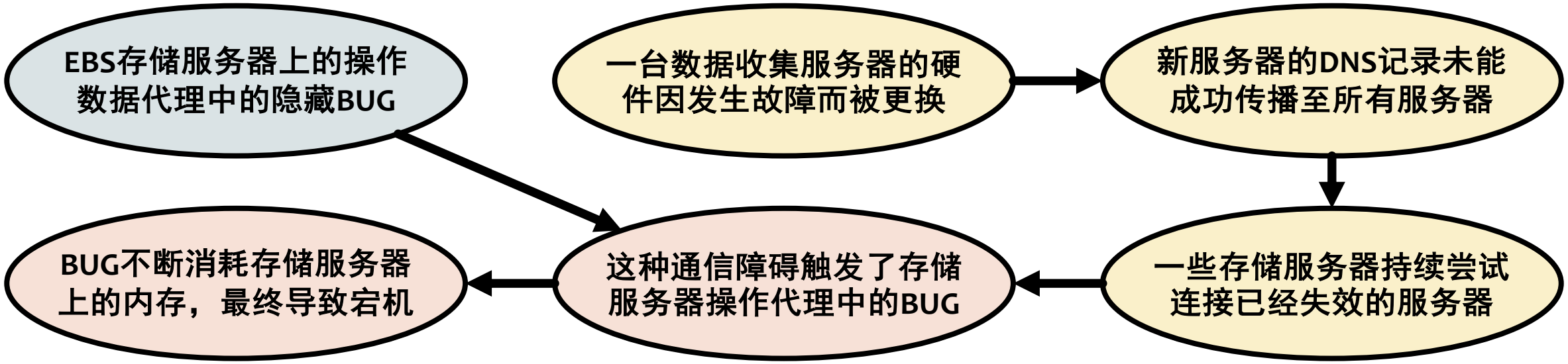
应用感知难点三：多问题并存

- 多个原因导致一个应用运行问题 / 多个应用运行问题同时存在

Summary of the October 22, 2012 AWS Service Event in the US-East Region

We'd like to share more about the service event that occurred on Monday, October 22nd in the US- East Region. We have now completed the analysis of the events that affected AWS customers, and we want to describe what happened, our understanding of how customers were affected, and what we are doing to prevent a similar issue from occurring in the future.

The Primary Event and the Impact to Amazon Elastic Block Store (EBS) and Amazon Elastic Compute Cloud (EC2)



什么是好的应用感知结果

- 线性性 (*Linearity*)
  - ✓ 符合人的直觉
- 可靠性
  - ✓ 当A的应用感知结果优于B时，则应用A的运行状态总是好于应用B
- 可重复性
- 易用性
- 一致性
- 独立性

应用感知技术：基本原则与分类

- 应用感知技术基本原则：**先整体、再局部**

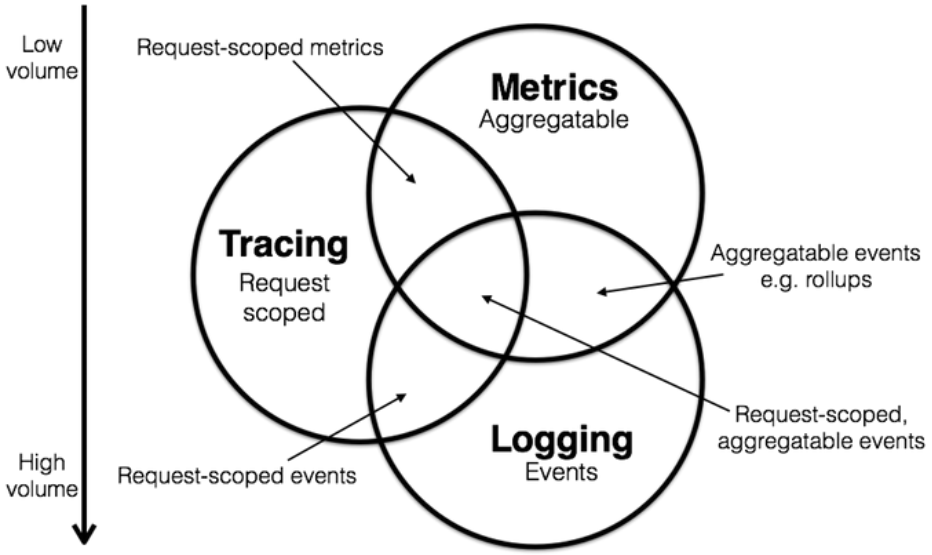
- **可观测性**应用感知技术
  - ✓ 三基石：*Metrics* (性能指标)、*Tracing* (事件轨迹)、*Logging* (运行日志)
  - ✓ *Profiling* (程序剖面)

- 非可观测性应用感知技术
  - ✓ *Simulation* (模拟)、*Emulation* (仿真)、*Testing* (测试)

应用感知技术：可观测性

- 可观测性定义 (*Control Theory*)
  - ✓ *A measure of how well internal states of a system can be inferred from knowledge of its external outputs. (Rudolf E. Kálmán, 1960)*

- *Observability Three Pillars*
  - ✓ *Metrics* (性能指标)
  - ✓ *Tracing* (事件轨迹)
  - ✓ *Logging* (运行日志)



可观测性应用感知技术：*Metrics* (性能指标)

- 延时 (*Latency*)
  - ✓ 所有操作完成的耗时
  - ✓ 一次应用程序请求、一次数据库查询、一次文件系统操作
- 操作计数、字节计数、资源使用率、硬件指标、错误率

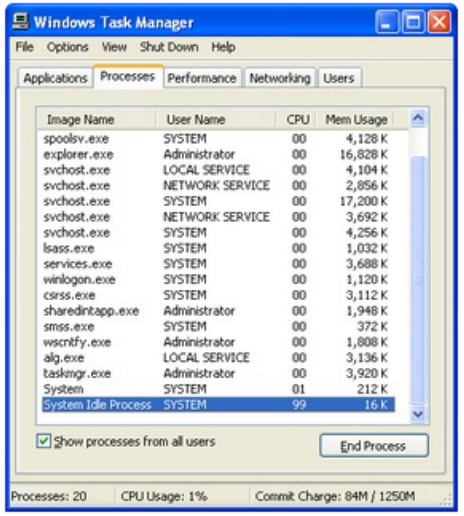
```
$ vmstat 1 3
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 4  0      0 34455620 111396 13438564   0    0     0     5    1    2  0  0 100  0
 4  0      0 34458684 111396 13438588   0    0     0     0 2223 15198 13 11 76  0
 4  0      0 34456468 111396 13438588   0    0     0     0 1940 15142 15 11 74  0
```

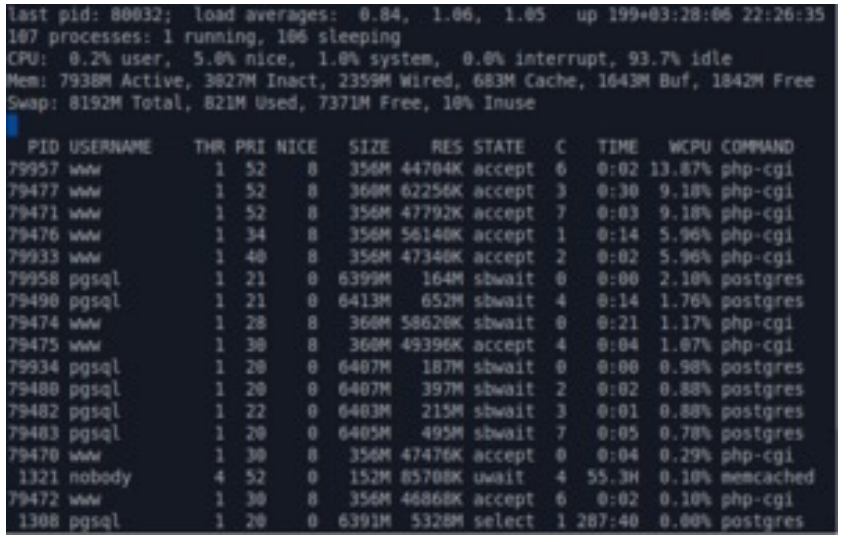
- 代表工具
  - ✓ *Unix (top、fuser、vmstat)*、*Windows* (任务管理器)
  - ✓ *Cacti*、*Nagios*、*Zabbix*
  - ✓ *Prometheus*

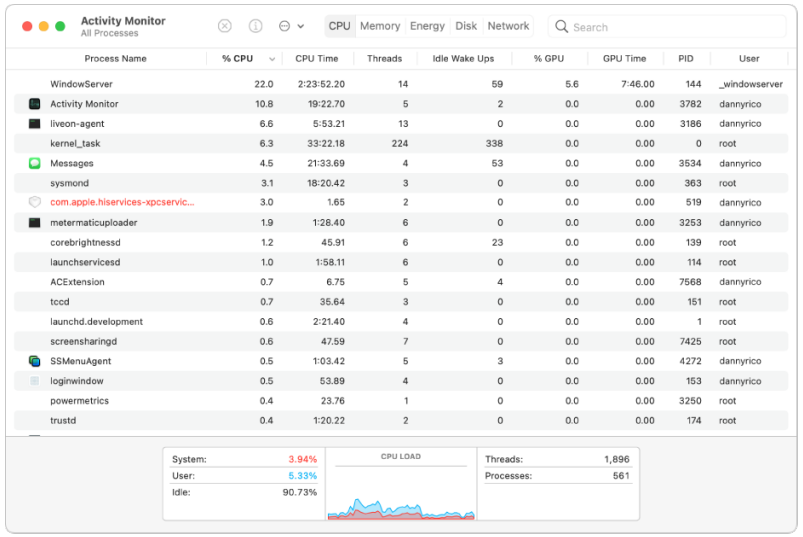
可观测性应用感知技术：Metrics (性能指标)

代表工具

- ✓ *Unix (top、fuser、vmstat)、Windows (任务管理器)*
- ✓ *Cacti、Nagios、Zabbix*
- ✓ *Prometheus*



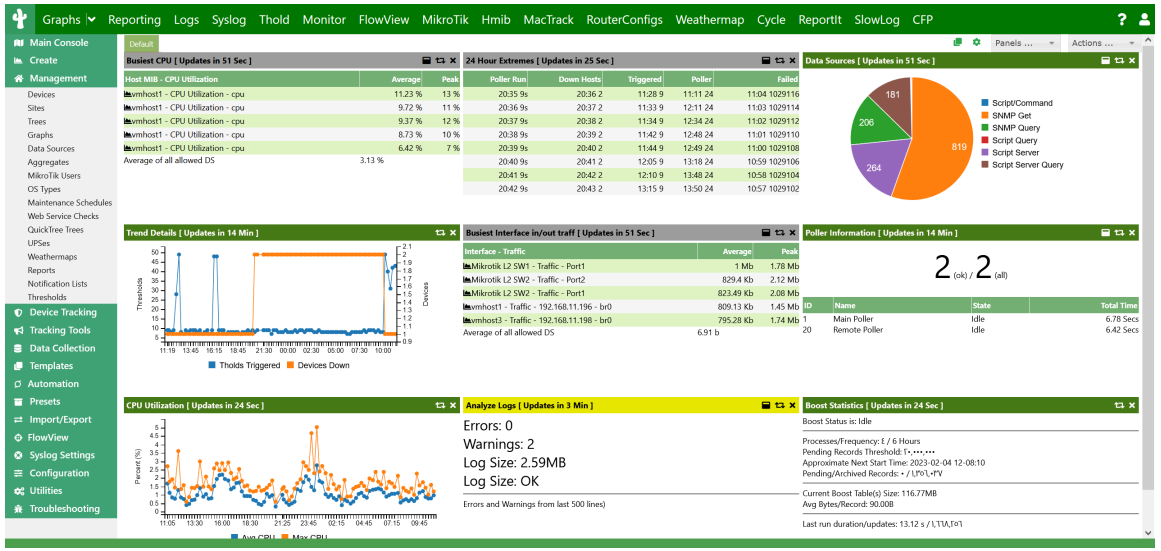


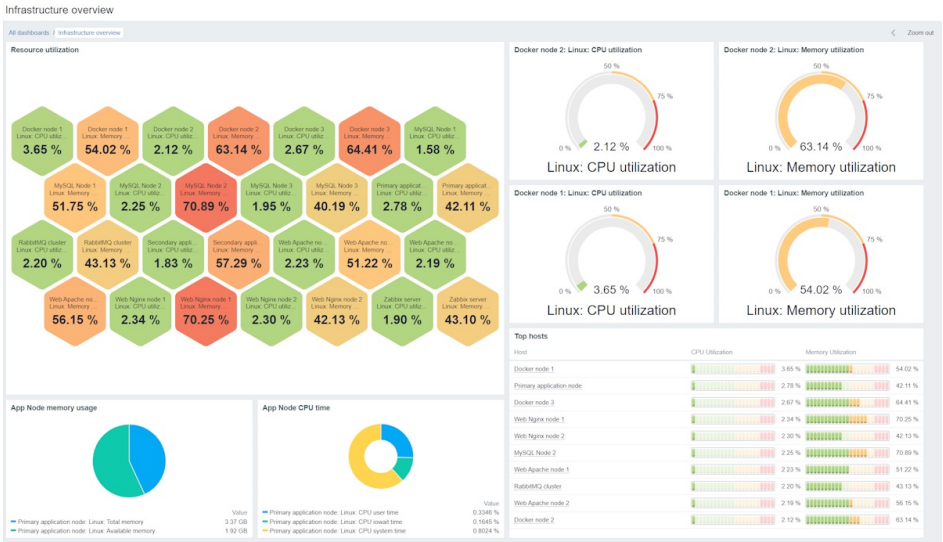


可观测性应用感知技术：*Metrics* (性能指标)

◻ 代表工具

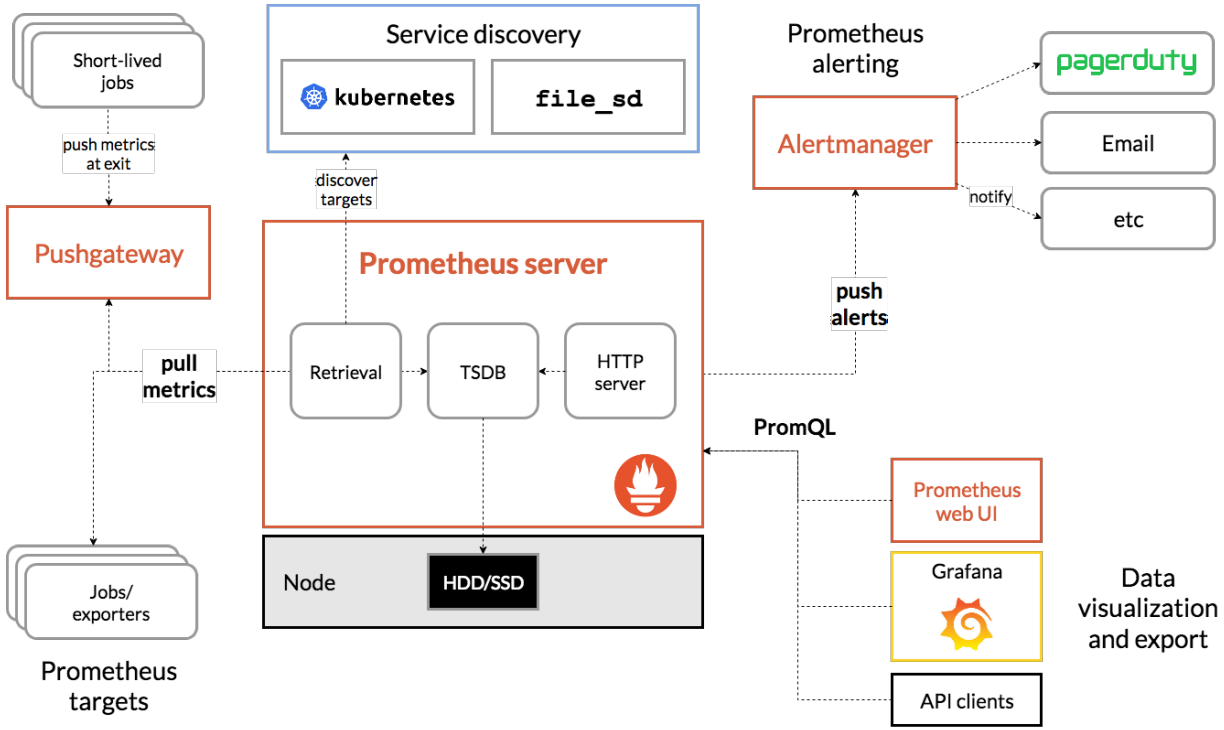
- ✓ *Unix (top、fuser、vmstat)、Windows (任务管理器)*
- ✓ *Cacti、Nagios、Zabbix*
- ✓ *Prometheus*





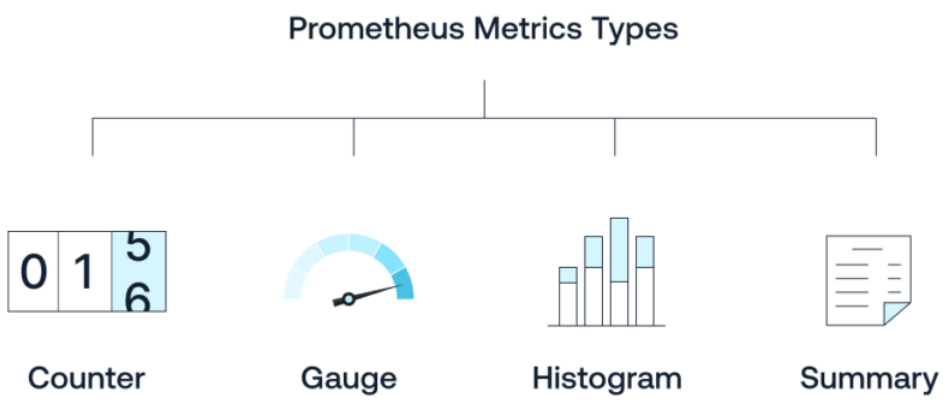
可观测性应用感知技术：Metrics (性能指标)

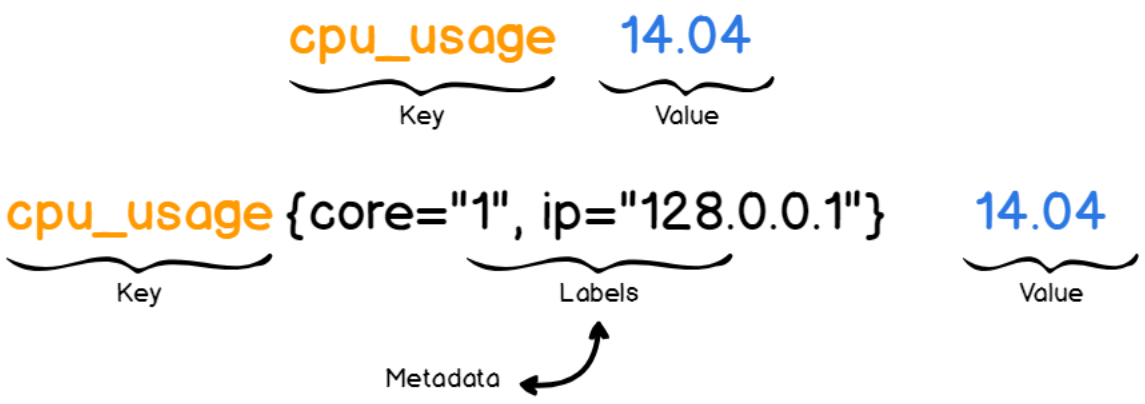
- 代表工具
  - ✓ Unix (top、fuser、vmstat)、Windows (任务管理器)
  - ✓ Cacti、Nagios、Zabbix
  - ✓ Prometheus
- Prometheus Components
  - ✓ the main Prometheus server
  - ✓ client libraries
  - ✓ a push gateway
  - ✓ special-purpose exporters
  - ✓ an alert manager
  - ✓ various support tools



可观测性应用感知技术：*Metrics* (性能指标)

- ***Metric Types in Prometheus***
  - ✓ ***Counter、Gauge、Histogram、Summary***
- ***Data Model***
  - ✓ ***Key、Value、Metadata***





可观测性应用感知技术：*Tracing* (事件轨迹)

事件发生的顺序信息对感知应用状态是必要的

程序访存序列1：

A	A	B	B	C	C	D	D

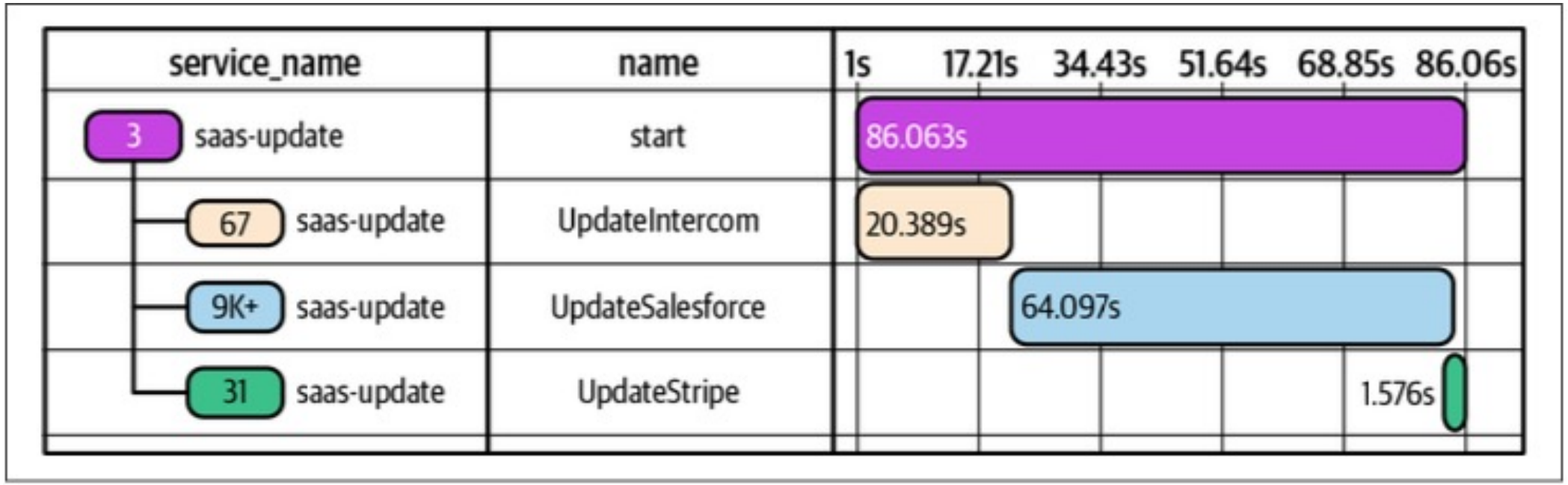
程序访存序列2：

A	B	C	D	A	B	C	D

问题：对于cache容量为3的，采用LRU替换策略，两个程序的cache缺失次数一样吗？

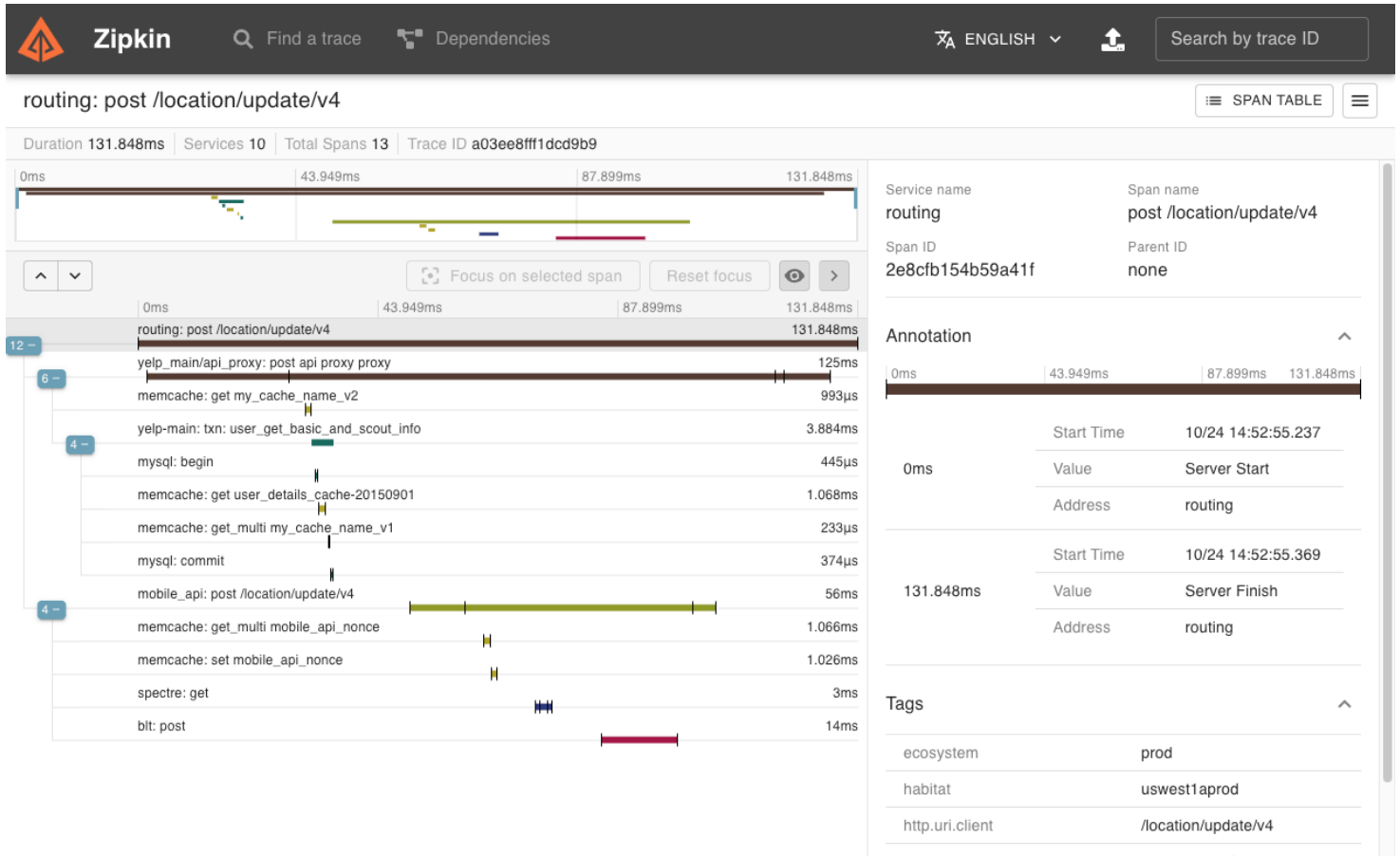
可观测性应用感知技术：*Tracing* (事件轨迹)

- ***Traces***
  - ✓ 按照应用执行的时间顺序记录每条事件的详细信息
- ***Distributed Tracing***



可观测性应用感知技术：*Tracing* (事件轨迹)

- 代表工作
  - ✓ *Dapper* (稍后介绍)
  - ✓ *Zipkin*、*Jaeger*
- *Twitter*
  - ✓ *BigBrotherBird, 2011*
  - ✓ *OpenZipkin, 2015*
  - ✓ *Scala - Java*



可观测性应用感知技术：*Tracing* (事件轨迹)

- 代表工作
  - ✓ *Dapper* (稍后介绍)
  - ✓ *Zipkin*、*Jaeger*

JAEGER

All in One

all-in-one is an executable designed for quick local testing. It includes the Jaeger UI, **jaeger-collector**, and **jaeger-query**, with an in memory storage component.

The simplest way to start the all-in-one is to use the pre-built image published to DockerHub (a single command line).

Copy

```
docker run --rm --name jaeger \
  -e COLLECTOR_ZIPKIN_HOST_PORT=:9411 \
  -p 16686:16686 \
  -p 4317:4317 \
  -p 4318:4318 \
  -p 14250:14250 \
  -p 14268:14268 \
  -p 14269:14269 \
  -p 9411:9411 \
  jaegertracing/all-in-one:1.63.0
```

Or run the `jaeger-all-in-one(.exe)` executable from the binary distribution archives:

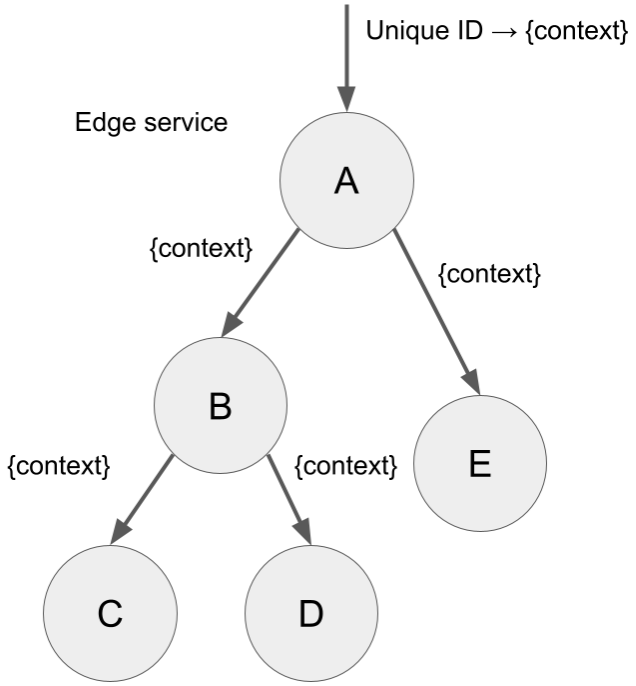
Copy

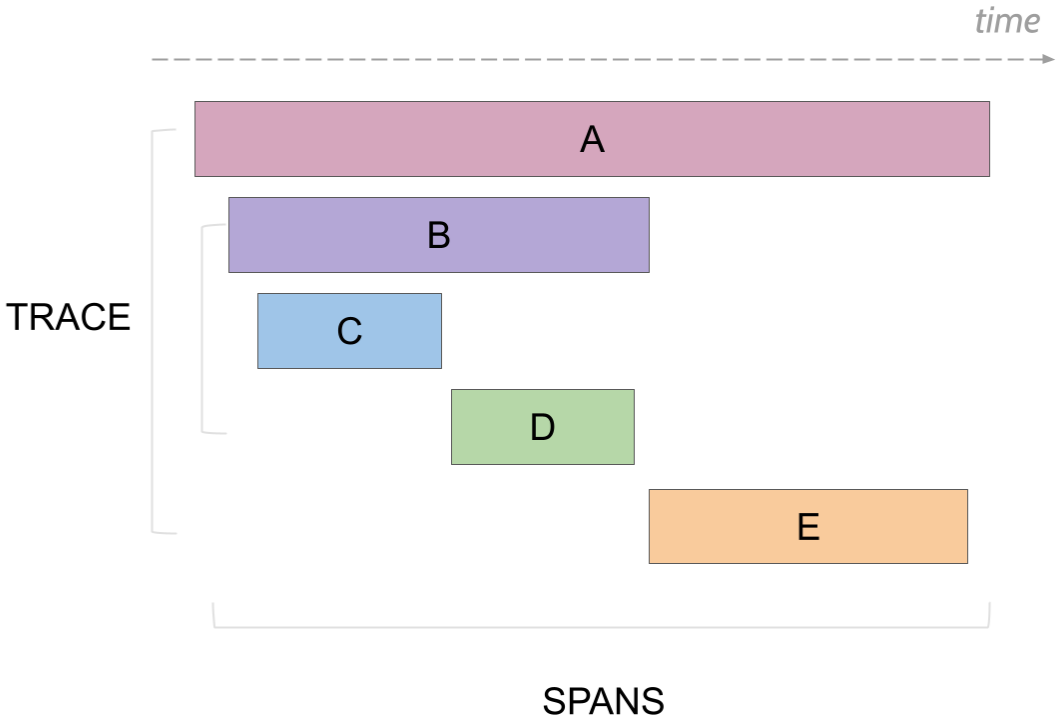
```
jaeger-all-in-one --collector.zipkin.host-port=:9411
```

可观测性应用感知技术：*Tracing* (事件轨迹)

☐ *Data Model*

✓ ***Span*、*Context*、*Traces***





可观测性应用感知技术： *Logging* (运行日志)

- 运行日志
  - ✓ 大量的非结构化文本块，设计为人类可读
  - ✓ 由应用程序和各种底层基础设施系统生成的文档，包含了所有已发生的重要事件的记录

可观测性应用感知技术： *Logging* (运行日志)

- 运行日志
  - ✓ 大量的非结构化文本块，设计为人类可读
  - ✓ 由应用程序和各种底层基础设施系统生成的文档，包含了所有已发生的重要事件的记录
- 非结构化日志

```
6:01:00 accepted connection on port 80 from 10.0.0.3:63349
6:01:03 basic authentication accepted for user foo
6:01:15 processing request for /super/slow/server
6:01:18 request succeeded, sent response code 200
6:01:19 closed connection to 10.0.0.3:63349
```

可观测性应用感知技术： *Logging* (运行日志)

- 运行日志
  - ✓ 大量的非结构化文本块，设计为人类可读
  - ✓ 由应用程序和各种底层基础设施系统生成的文档，包含了所有已发生的重要事件的记录
- 非结构化日志

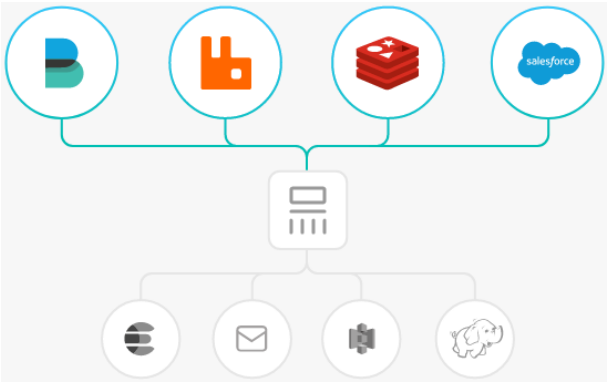
```
6:01:00 accepted connection on port 80 from 10.0.0.3:63349
6:01:03 basic authentication accepted for user foo
6:01:15 processing request for /super/slow/server
6:01:18 request succeeded, sent response code 200
6:01:19 closed connection to 10.0.0.3:63349
```

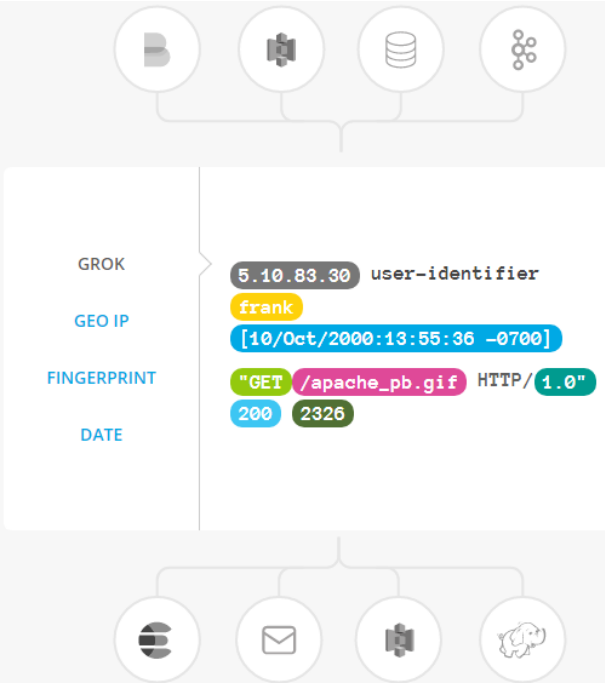
- 结构化日志

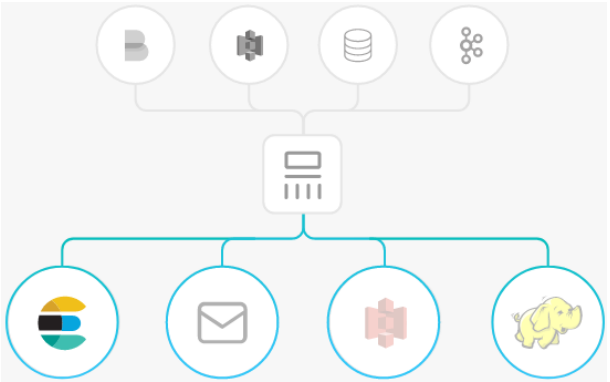
```
time="6:01:00" msg="accepted connection" port="80" authority="10.0.0.3:63349"
time="6:01:03" msg="basic authentication accepted" user="foo"
time="6:01:15" msg="processing request" path="/super/slow/server"
time="6:01:18" msg="sent response code" status="200"
time="6:01:19" msg="closed connection" authority="10.0.0.3:63349"
```

可观测性应用感知技术： *Logging* (运行日志)

- 代表工作
  - ✓ *Logstash*、*Fluentd*







输入： 采集各种样式、大小和来源的数据

过滤器： 实时解析和转换数据

输出： 选择你的存储，导出你的数据

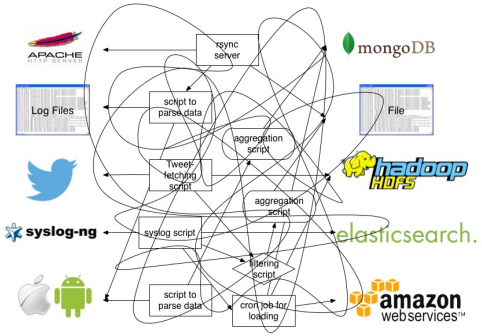
可观测性应用感知技术： *Logging* (运行日志)

◻ 代表工作

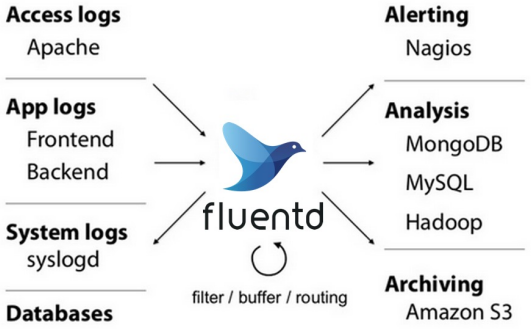
- ✓ ***Logstash***、***Fluentd***

◻ 主要特性

- ✓ ***Unified Logging***、***Pluggable Architecture***
- ✓ ***Minimum Resources***、***Built-in Reliability***



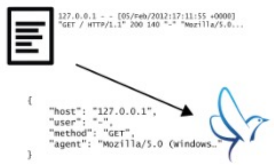
Before Fluentd



After Fluentd

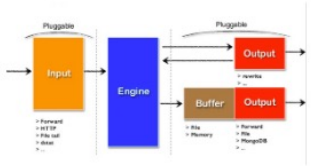
Fluentd is an open source data collector, which lets you unify the data collection and consumption for a better use and understanding of data.

Here're the key features.



Unified Logging with JSON

Fluentd tries to structure data as JSON as much as possible: this allows Fluentd to **unify** all facets of processing log data: collecting, filtering, buffering, and outputting logs across **multiple sources and destinations** (Unified Logging Layer). The downstream data processing is much easier with JSON, since it has enough structure to be accessible while retaining flexible schemas.

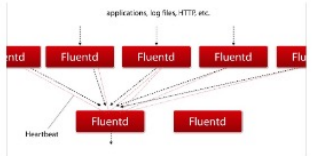


Pluggable Architecture

Fluentd has a flexible plugin system that allows the community to extend its functionality. Our 500+ community-contributed plugins connect dozens of data sources and data outputs. By leveraging the plugins, you can start making better use of your logs right away.

Minimum Resources Required

Fluentd is written in a combination of C language and Ruby, and requires very little system resource. The vanilla instance runs on 30-40MB of memory and can process 13,000 events/second/core. If you have tighter memory requirements (-450kb), check out Fluent Bit, the lightweight forwarder for Fluentd.

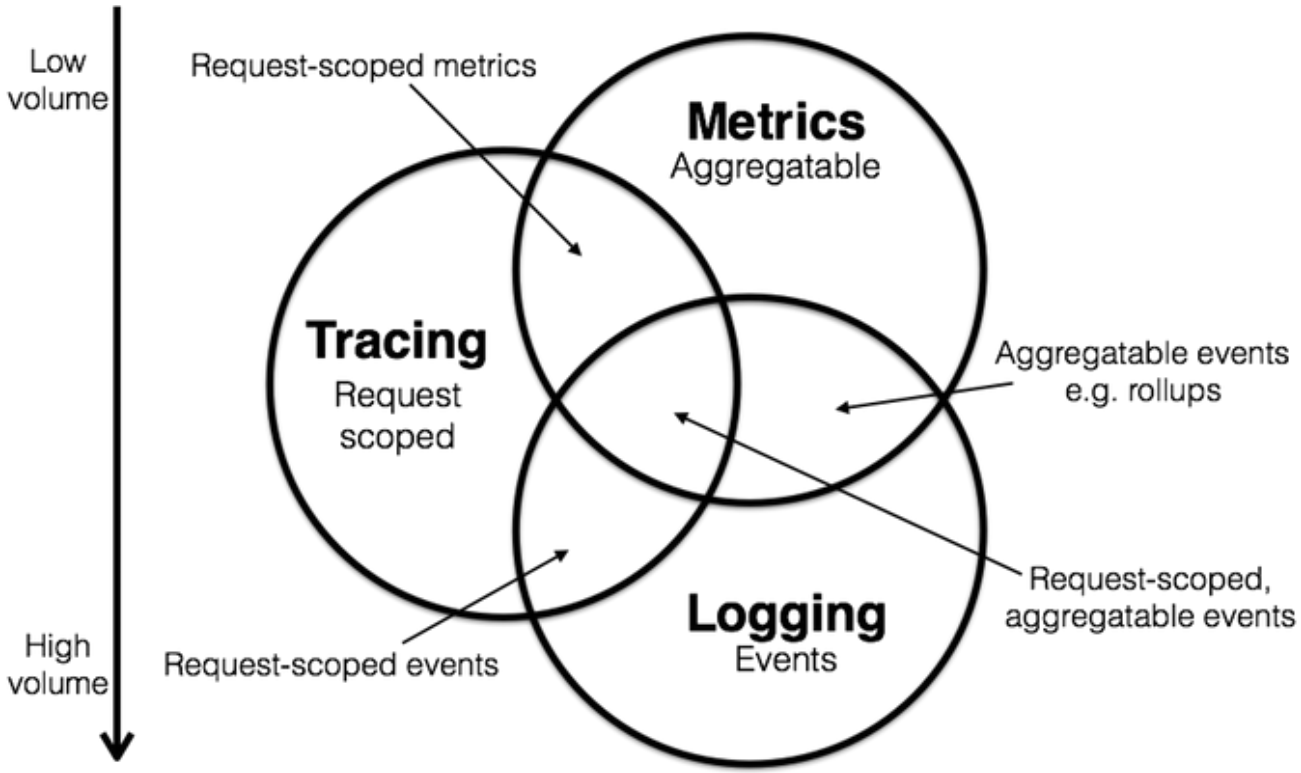


Built-in Reliability

Fluentd supports memory- and file-based buffering to prevent inter-node data loss. Fluentd also supports robust failover and can be set up for high availability. 2,000+ data-driven companies rely on Fluentd to differentiate their products and services through a better use and understanding of their log data.

应用感知技术：可观测性

- *Observability Three Pillars*
  - ✓ *Metrics* (性能指标)
    - ◆ *Aggregatable*
  - ✓ *Tracing* (事件轨迹)
    - ◆ *Request scoped*
  - ✓ *Logging* (运行日志)
    - ◆ *Events*

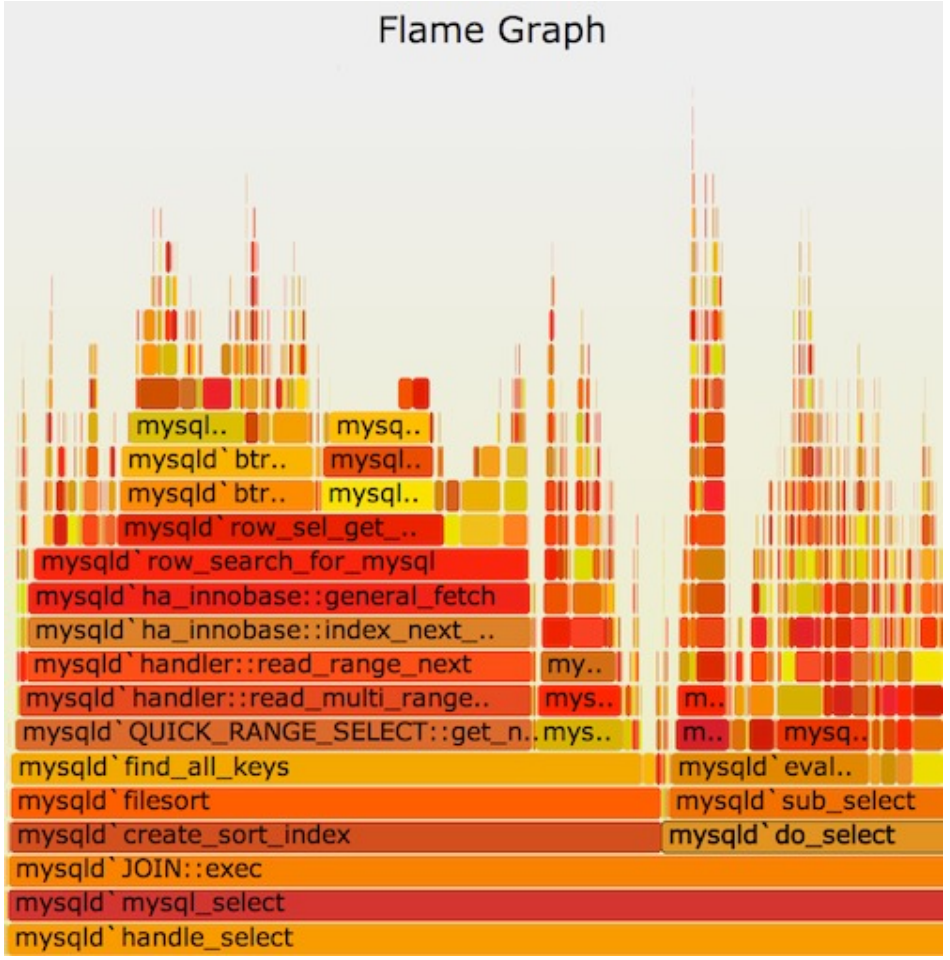


应用感知技术不只是单纯的数据收集，更是一种确保系统可靠性和效率的方法论

可观测性应用感知技术： *Profiling* (程序剖面)

- *Profiling*
  - ✓ 一个应用程序执行行为的统计信息
  - ✓ 例子：人事部门统计销售人员8月份的出差信息
    - ◆ 事件轨迹 (*Trace*)：
      - ➢ 8月5日(周五)18:00 ➔ 8月7日(周日)21:00　长沙
      - ➢ 8月13日(周六)18:00 ➔ 8月16(周二)22:00　武汉
      - ➢ 8月20日(周六)20:00 ➔ 8月22(周一)19:00　哈尔滨
    - ◆ 程序剖面 (*Profile*)：
      - ➢ 武汉3天，长沙2天，哈尔滨2天

- 火焰图 (*Flame Graph*)
  - ✓ 一组水平条形图
  - ✓ 每个条形图都代表了应用栈中的一个事务调用
  - ✓ 宽度表示该事务调用在总执行时间中的比例

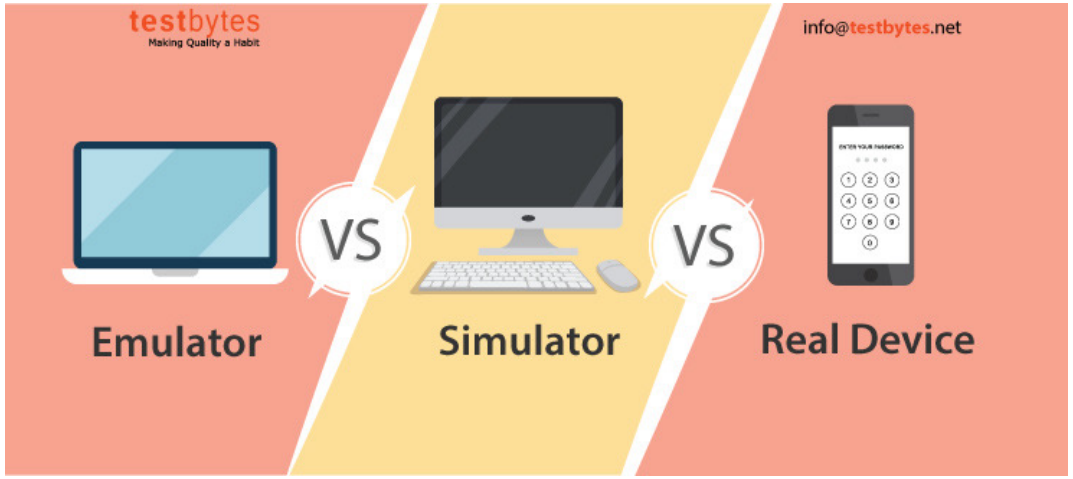


非可观测性应用感知技术

- **Simulation (模拟)**
  - ✓ **创建一个被感知应用的行为模型，通过这种方式来预测应用在各种条件下的运行状态**
    - ◆ 应用运行特征 (收集浮点指令、整数指令、访存记录、I/O访问、通信等)；机器性能 (每条浮点/整数指令执行的*cycle*，访存指令时间、I/O性能)；通信延迟、带宽；程序执行时间
  - ✓ 例子：数学建模、排队论

- **Emulation (仿真)**
  - ✓ **通过硬件或软件解决方案在一个系统上模拟另一个系统的应用运行环境**
    - ◆ 应用运行特征、目标系统的参数
  - ✓ 例子：功能模拟器、性能模拟器、全系统模拟

- **Testing (测试)**
  - ✓ **在实际的或与实际情况尽可能相似的环境中验证应用**
    - ◆ 工作负载特征、应用运行特征
  - ✓ 例子：Micro-Benchmarks、Benchmarking



	Technique		
	Simulation	*Emulation*	*Testing*
Flexibility	*High*	*High*	*Low*
Cost	*Low*	*Medium*	*High*
Believability	*Low*	*Medium*	*High*
Accuracy	*Low*	*Medium*	*High*

目录

- 应用感知技术介绍
- 应用感知目标、难点与技术
- **应用感知插桩与BPF技术**
- 应用感知技术相关研究

*Every application has an inherent amount of irreducible complexity. The only question is: who will have to deal with it—**the user, the application developer, or the platform developer**?*

— Larry Tesler, 1984

插桩

- 插桩 (*Instrumentation*)
  - ✓ 向应用程序代码中添加额外的代码或使用特定工具，在应用运行时收集关于程序行为和性能的数据

```
if (b > c)
  t = 1;
else
  b = 3;
```

```
if (b > c) {
    some_counter[0]++;
    t = 1;
} else {
    some_counter[1]++;
    b = 3;
}
```

插桩代码

插桩

- 插桩可以在不同阶段完成：
  - ✓ 源码插桩
  - ✓ 编译插桩
  - ✓ 链接插桩 (*LD_PRELOAD*)
    - ◆ 动态链接器会先读取LD_PRELOAD环境变量
    - ◆ 它们的优先级比LD_LIBRARY_PATH环境变量所定义的链接库查找路径的文件优先级要高
    - ◆ 能够提前于用户调用的动态库载入
  - ✓ 运行插桩 (*PIN*)
    - ◆ Intel和Virginia大学开发
    - ◆ http://www.pintool.org/
    - ◆ 免费，但不开源

Pin - A Binary Instrumentation Tool - Papers

Submitted by Sion Berkowits ... on Thu, 08/09/2012 - 02:58

Home | News | Downloads | FAQ | Newsgroup | Papers | PinPoints

A small subset of the conference publications about Pin or using Pin. The text for most of the entries below was created by going to the ACM Library, finding the paper, clicking on 'ACM Ref' and doing a cut and paste. If there is an ACM link for a reference, click through to get the full text or a bibtex reference.

For an up-to-date list of all papers citing the primary Pin paper, see Google Scholar.

2012

- Gregory Lueck, Harish Patil, Cristiano Pereira. "**PinADX: An Interface for Customizable Debugging with Dynamic Instrumentation**". Proceedings of the IEEE/ACM international symposium on Code generation and optimization (CGO 2012), pages 114-123. slides. DOI.

2011

- Amitabha Roy, Steven Hand, and Tim Harris. "**Hybrid Binary Rewriting for Memory Access Instrumentation**". Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '11), pages 227-238. DOI

2010

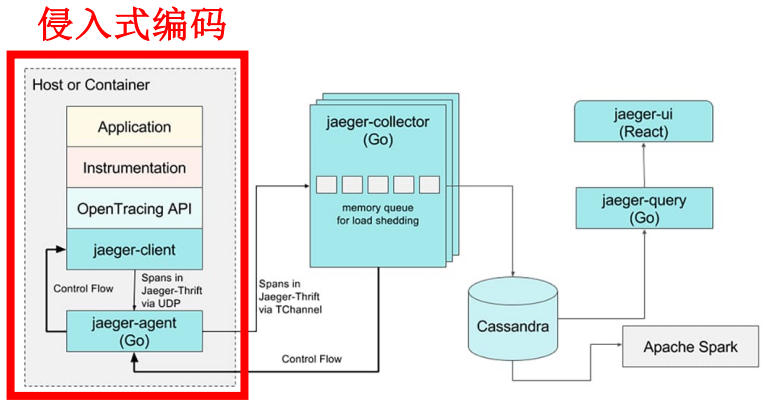
- Alex Skaletsky, Tevi Devor, Nadav Chachmon, Robert Cohn, Kim Hazelwood, Vladimir Vladimirov, Moshe Bach. "**Dynamic Program Analysis of Microsoft Windows Applications**," International Symposium on Performance Analysis of Software and Systems (ISPASS). White Plains, NY. April 2010.
- Harish Patil, Cristiano Pereira, Mack Stallcup, Gregory Lueck, James Cownie. "**PinPlay: A Framework for Deterministic Replay and Reproducible Analysis of Parallel Programs**". *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization (CGO 2010).* **CGO 2010 Best Paper Award Winner.** DOI
- Moshe Bach, Mark Charney, Robert Cohn, Tevi Devor, Elena Demikovsky, Kim Hazelwood, Aamer Jaleel, Chi-Keung Luk, Gail Lyons, Harish Patil, Ady Tal. "**Analyzing Parallel Programs with Pin**," IEEE Computer. March 2010, vol. 43, no. 3, pages 34-41.

关于是否插桩……

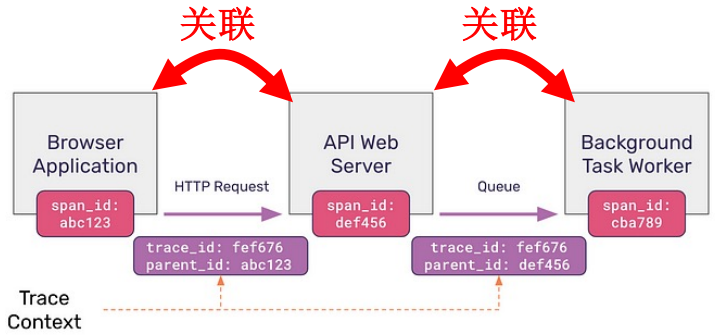
◻ 以应用为中心的应用感知

- ✓ 在应用运行前，通过编程手段，将监测行为以侵入式方式硬编码至应用中
  - ◆ 监测逻辑为分布式应用的一部分
  - ◆ 采集并分析应用级别的监控数据

- ✓ 代表性工作
  - ◆ 分布式追踪框架
    - ➢ Jaeger、Zipkin、OpenTelemetry等
- ✓ 优点：
  - ◆ 应用感知精确度高
- ✓ 缺点：监控行为与分布式应用紧密耦合
  - ◆ 应用感知成本高



以应用为中心的分布式应用感知需要通过插桩 (Instrumentation) 方式，侵入式编码监控逻辑，并调用监控客户端提供的监控API



以应用为中心的分布式应用感知能够修改用户请求，以支撑请求的端到端观测

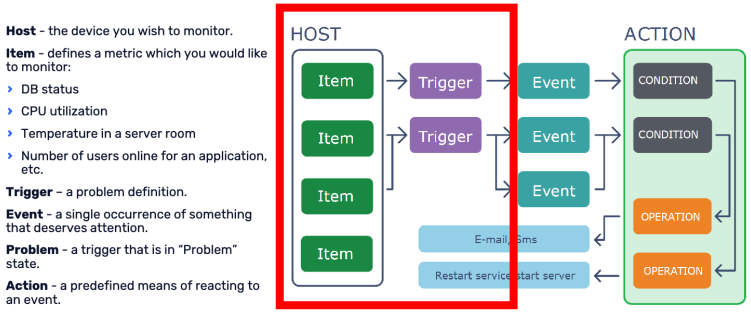
关于是否插桩……

□ 以系统为中心的应用感知

✓ 在应用运行后，观测基础设施的运行状态，以此推断风险/故障/安全事件

- ◆ 通过监测基础设施，间接监测应用
- ◆ 采集并分析系统级别的监控数据

监控Host运行状态



以系统为中心的分布式应用感知对每个计算节点的关键性能指标进行监控

✓代表性工作

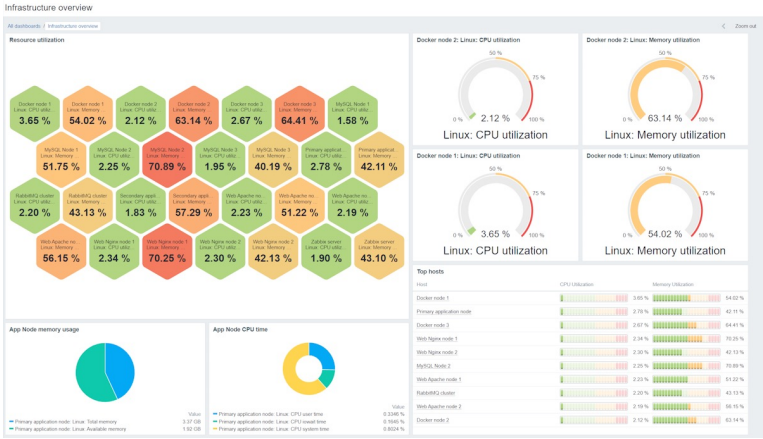
- ◆ 云基础设施监控系统
  - ➢ Zabbix、Nagios、Icinga等

✓ 优点：

- ◆ 应用感知成本低

✓ 缺点：无法直接获取应用运行状态

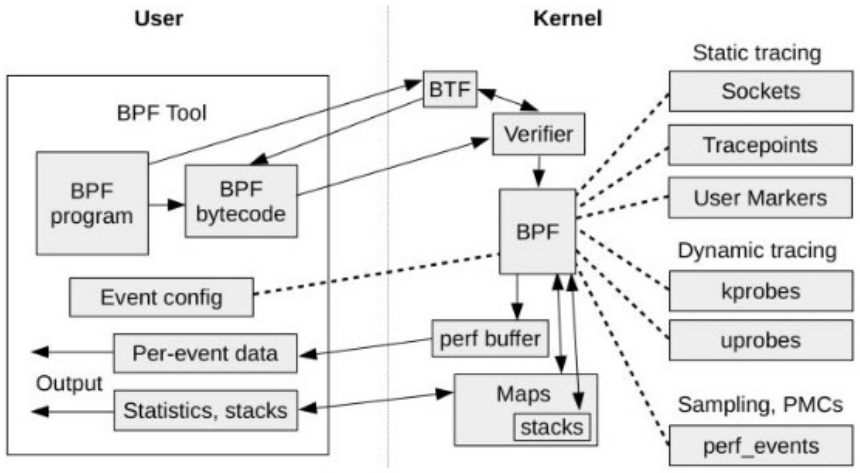
- ◆ 应用感知精度低

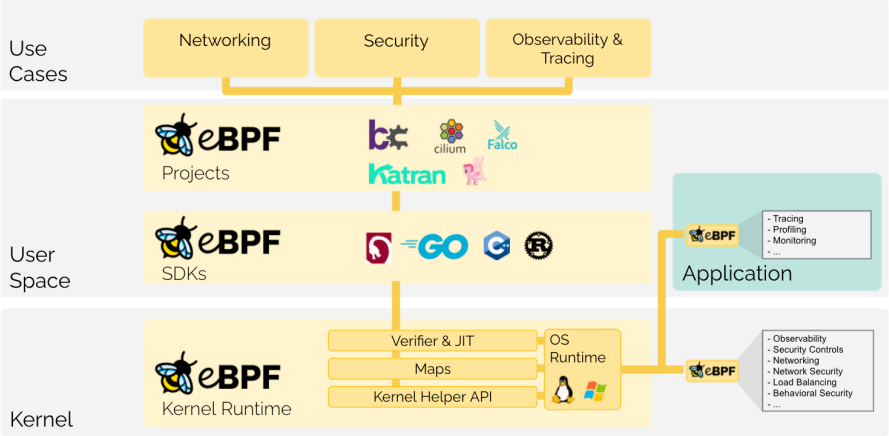


以系统为中心的分布式应用感知：云基础设施监控大盘

也许我们有更好的选择……

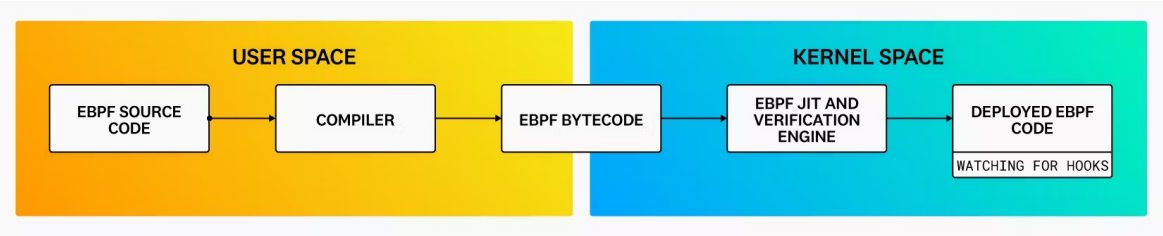
- **Berkeley Packet Filter (BPF)**
  - ✓ *Improved the performance of packet capture tools (1992)*
  - ✓ *Major rewrite (Alexei Starovoitov, Daniel Borkmann, 2013)*
  - ✓ *Included in the Linux kernel (2014)*
- ***What BPF can do is …***
  - ✓ *a way to run mini programs on a wide variety of kernel and application events*
  - ✓ *Just like JavaScript*

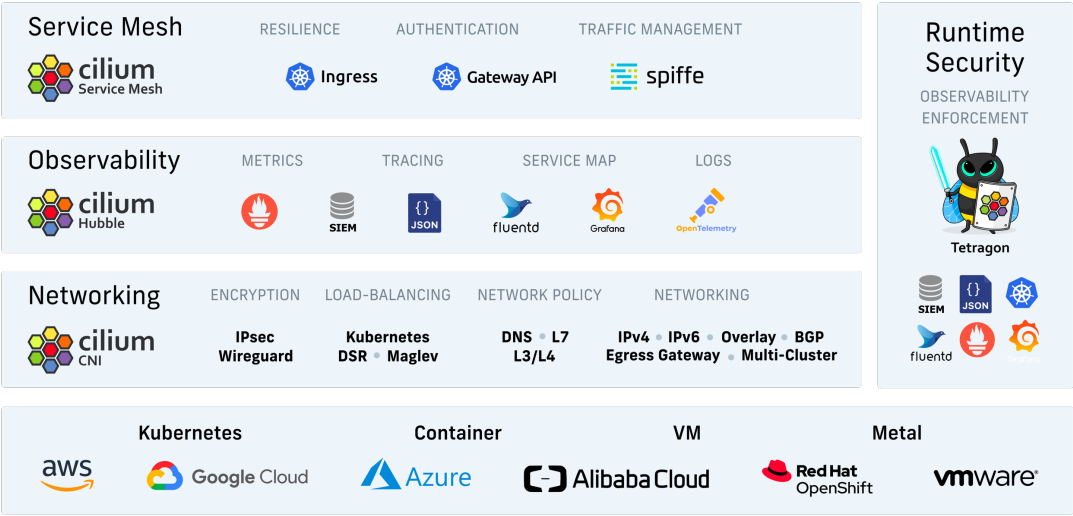


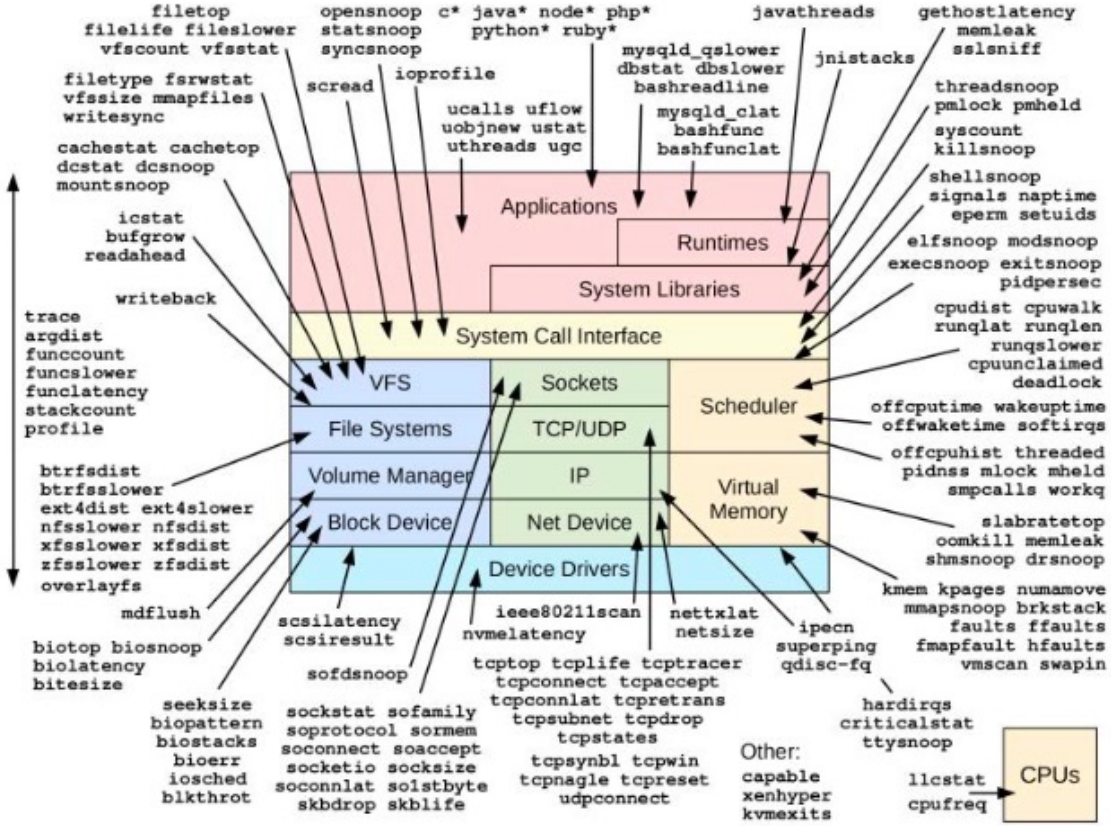


也许我们有更好的选择……

□ BPF 极大扩充了应用感知技术







总结

- 随着现代应用的功能、规模与复杂性不断提升，故障修复、性能优化、调度决策、安全防护等任务需要应用感知技术的支撑
- 应用运行状态是主观、复杂的，且可能同时并存多个问题，应用感知技术不仅需要感知应用“已知的已知”、“已知的未知”，还需要感知“未知的未知”
- 应用感知本质上是通过可记录状态（性能指标、事件轨迹、运行日志）描述的应用状态。因此很难把无限的应用状态运行空间映射到有限维度的空间上，用一些简单的数据结构描述应用的运行状态，即绝对可靠的应用感知技术不存在
- 插桩是应用感知技术的最大痛点，*eBPF*技术的出现极大降低了应用感知的实现成本，扩展了应用感知的能力边界

目录

- 应用感知技术介绍
- 应用感知目标、难点与技术
- 应用感知插桩与BPF技术
- **应用感知技术相关研究**

Dapper, a Large-Scale Distributed Systems Tracing Infrastructure

Google Technical Report 2010

Benjamin H. Sigelman, Luiz Andr´ e Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag

Distributed Systems

- ***Modern Internet services are often implemented as complex, large-scale distributed systems.***

- ***These applications are constructed from collections of software modules that may be developed by different teams, perhaps in different programming languages, and could span many thousands of machines across multiple physical facilities.***

- ***Tools that aid in understanding system behavior and reasoning about performance issues are invaluable in such an environment***

An Example – Web Search

- ***A front-end service may distribute a web query to***
  - ✓ ***many hundreds of query servers, each searching within its own piece of the index***
  - ✓ ***a number of other sub-systems that may process advertisements, check spelling, or look for specialized results, including images, videos, news, and so on***
- ***In total, thousands of machines and many different services might be needed to process one universal search query.***
- ***An engineer looking only at the overall latency may know there is a problem, but may not be able to guess which service is at fault, nor why it is behaving poorly***
  - ✓ ***the engineer may not be aware precisely which services are in use***
  - ✓ ***the engineer will not be an expert on the internals of every service***
  - ✓ ***services and machines may be shared simultaneously by many different clients***

Fundamental Requirements

- ***Concepts***
  - ✓ ***Ubiquitous deployment***
  - ✓ ***Continuous monitoring***

- ***Specific Requirements***
  - ✓ ***Low overhead***
  - ✓ ***Application-level transparency***
    - ◆ ***programmers should not need to be aware of the tracing system***
  - ✓ ***Scalability***
  - ✓ ***Additional requirement: for tracing data to be available for analysis quickly after it is generated: ideally within a minute***

Distributed Tracing in Dapper

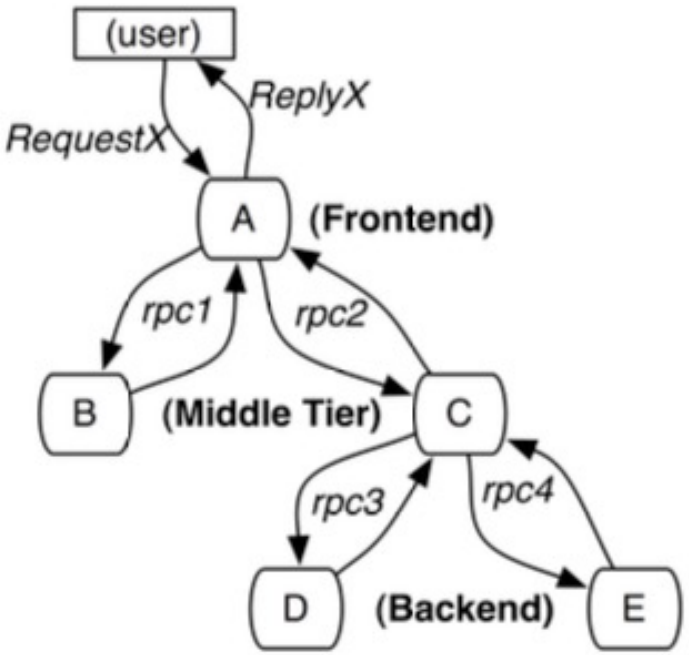


Figure 1: The path taken through a simple serving system on behalf of user request *X*. The letter-labeled nodes represent processes in a distributed system.

Trace Trees and Spans

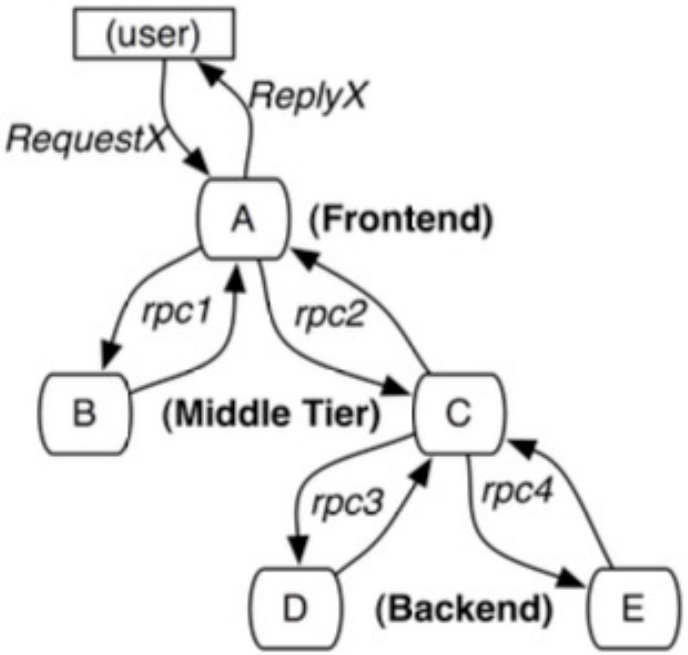


Figure 1: The path taken through a simple serving system on behalf of user request *X*. The letter-labeled nodes represent processes in a distributed system.

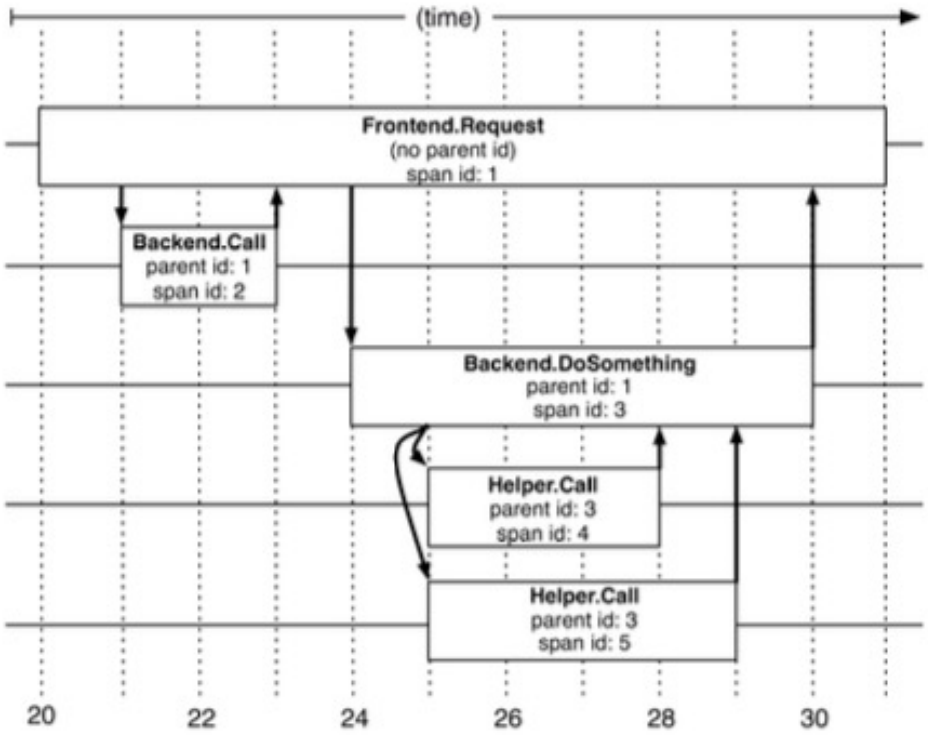


Figure 2: The causal and temporal relationships between five spans in a Dapper trace tree.

Spans

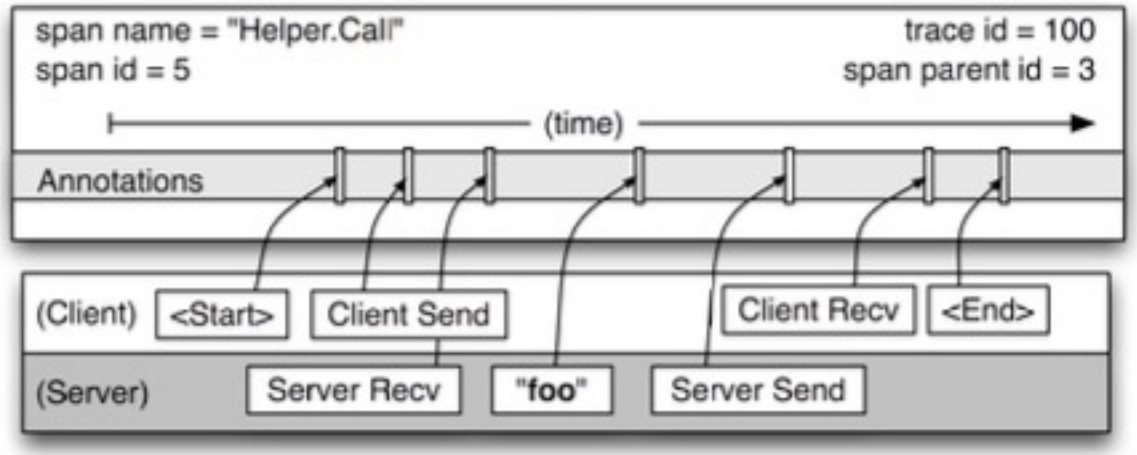


Figure 3: A detailed view of a single span from Figure 2.

Others (1)

- *Instrumentation points*
  - ✓ *trace context*
  - ✓ *deferred / asynchronous*
    - ◆ *a common control flow library*
    - ◆ *construct callbacks and schedule them in a thread pool or other executor*
  - ✓ *inter-process*
    - ◆ *a single RPC framework in C++ and Java*

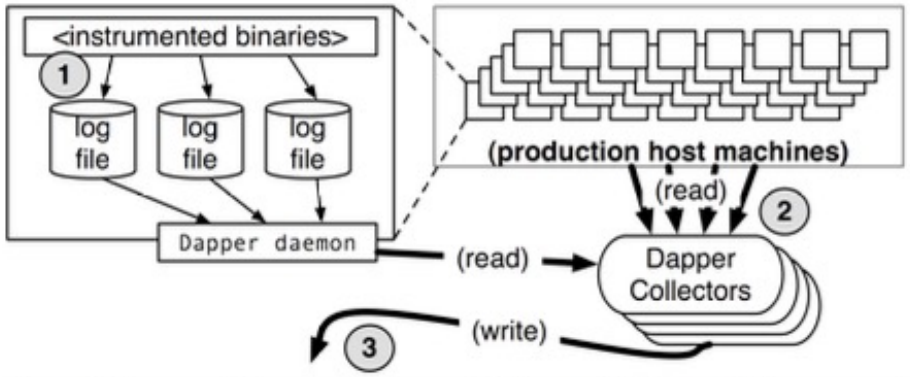
```
// C++:
const string& request = ...;
if (HitCache())
  TRACEPRINTF("cache hit for %s", request.c_str());
else
  TRACEPRINTF("cache miss for %s", request.c_str());

// Java:
Tracer t = Tracer.getCurrentTracer();
String request = ...;
if (hitCache())
  t.record("cache hit for " + request);
else
  t.record("cache miss for " + request);
```

Figure 4: Common-case usage patterns for Dapper's annotation APIs in C++ and Java.

Others (2)

- **Annotations**
  - ✓ *timestamped, textual annotations*
  - ✓ *a map of key-value annotations*
- **Sampling**
- **Trace collection**
  - ✓ *local log files*
  - ✓ *pulled by Dapper daemons and collection infra*
  - ✓ *written to a cell in one of several regional Dapper Bigtable repo*



trace id	span 12	span 23	span 34	span 45	span 56	...
123456	*nil*	*nil*	**<data>**	**<data>**	*nil*	...
246802	**<data>**	*nil*	*nil*	*nil*	**<data>**	...
357913	*nil*	**<data>**	*nil*	*nil*	*nil*	...
...	...	...	...	...	...	...

(Central Bigtable repository for trace data)

Dapper Deployment Status

- ***Dapper runtime library***
  - ✓ *less than 1000 lines of code in C++*
  - ✓ *under 800 lines in Java*
  - ✓ *key-value annotations adds an additional 500 lines of code*
- ***Production coverage – nothing new***
- ***Use of trace annotations***
  - ✓ *70% of all Dapper spans and 90% of all Dapper traces have at least one application-specified annotation*
- ***How about the current status of distributed tracing?***

Managing Tracing Overhead - FYI

- Trace generation overhead
  - ✓ root span creation and destruction - 204 ns
  - ✓ non-root spans - 176 ns
  - ✓ annotating the trace - 40 ns
- Trace collection overhead
  - ✓ daemon - <0.3% CPU core
  - ✓ network - 426 bytes
- Effect on production workloads
- Adaptive sampling – in the process
- Coping with aggressive sampling
  - ✓ If a notable execution pattern surfaces once in such systems, it will surface thousands of times

Sampling frequency	Avg. Latency (% change)	Avg. Throughput (% change)
1/1	16.3%	−1.48%
1/2	9.40%	−0.73%
1/4	6.38%	−0.30%
1/8	4.12%	−0.23%
1/16	2.12%	−0.08%
1/1024	−0.20%	−0.06%

Table 2: The effect of different [non-adaptive] Dapper sampling frequencies on the latency and throughput of a Web search cluster. The experimental errors for these latency and throughput measurements are 2.5% and 0.15% respectively.

Experiences

- ***More like usecases …***
  - ✓ ***Using Dapper during development***
  - ✓ ***Addressing long tail latency***
  - ✓ ***Inferring service dependencies***
  - ✓ ***Network usage of different services***
  - ✓ ***Layered and shared storage systems***
  - ✓ ***Firefighting with Dapper***
- ***Other lessons learned, more like limitations …***
  - ✓ ***Request buffering***
  - ✓ ***Batches, not requests***
  - ✓ ***Finding root causes***
  - ✓ ***Kernel-level info***

Canopy: An End-to-End Performance Tracing And Analysis System

SOSP 2017

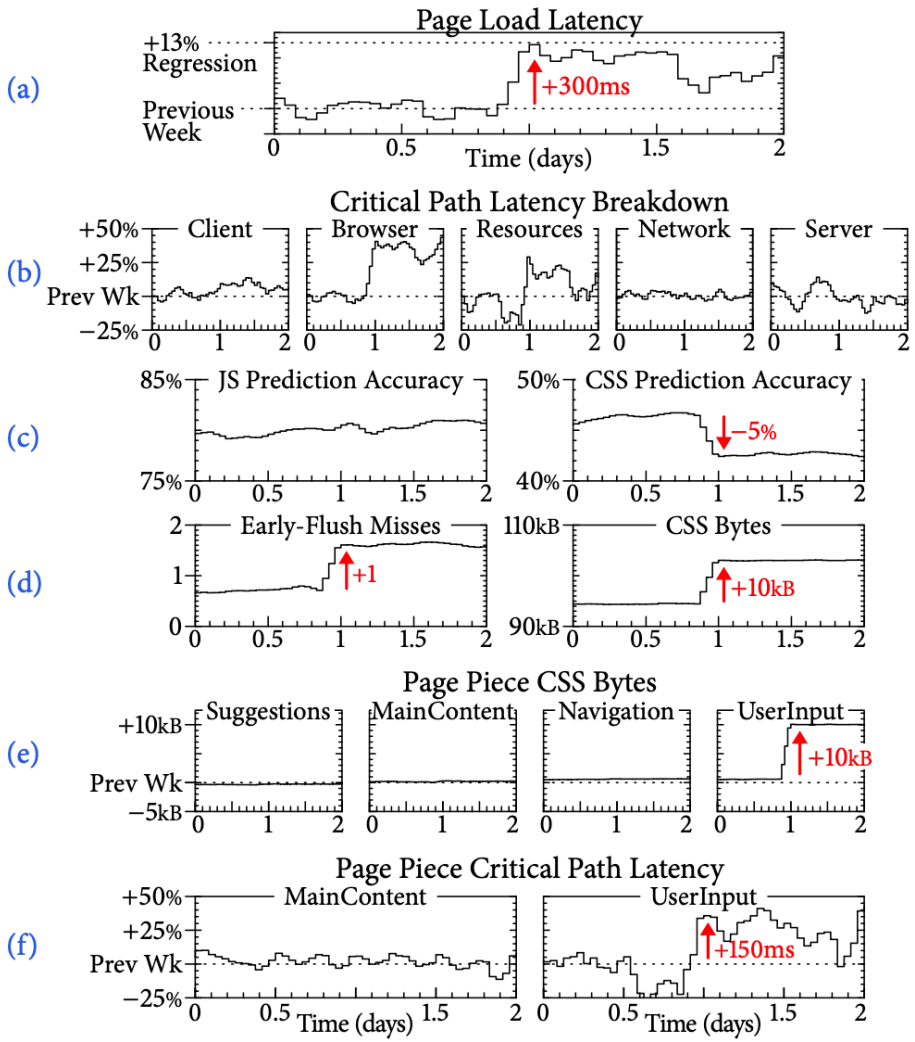
Jonathan Kaldor, Jonathan Mace, Michał Bejda, Edison Gao, Wiktor Kuropatwa,
JoeO'Neill, Kian Win Ong, Bill Schaller, Pingjia Shan, Brendan Viscomi,
Vinod Venkataraman, Kaushik Veeraraghavan, YeeJiun Song,
Facebook, Brown University

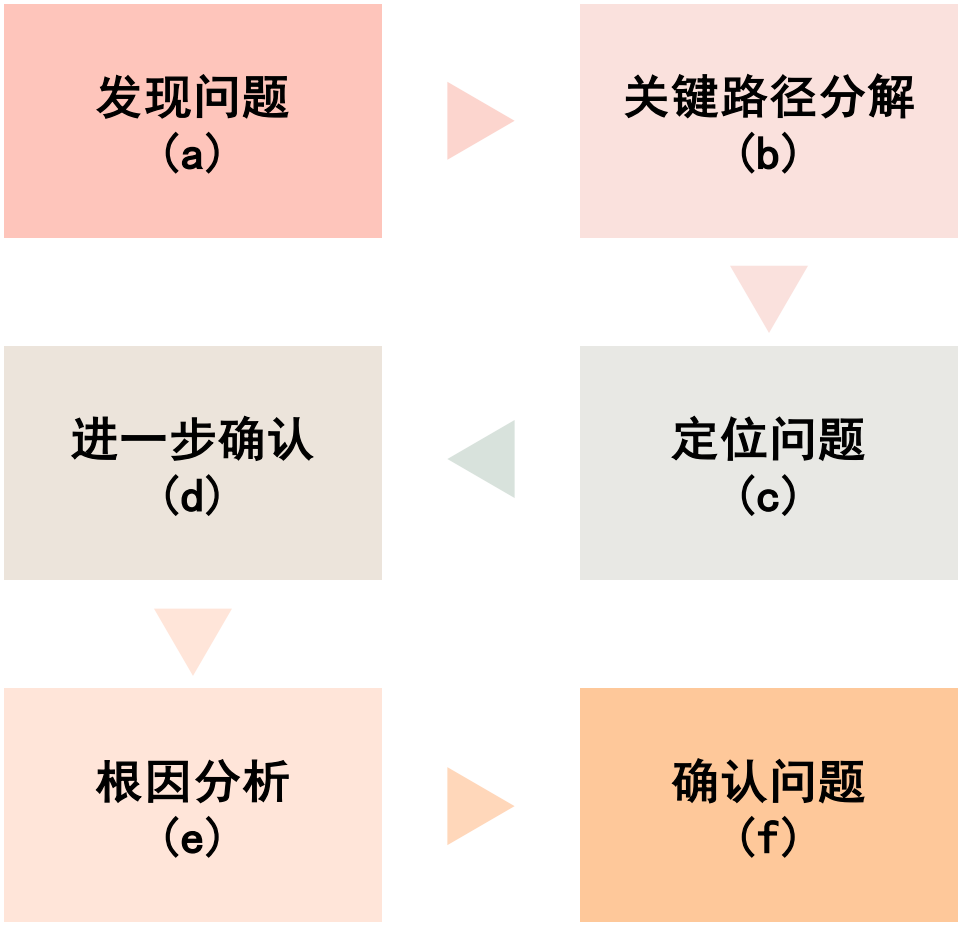
Challenges

Challenges

- 端到端的性能异质性（***Modeling Trace Data***）
  - ✓ 差异来源：执行模式/可供分析的数据粒度/质量
  - ✓ 现存问题
    - ◆ 通用性（队列、异步执行、多父因果关系）
    - ◆ 前后向兼容
    - ◆ 混合系统兼容性
- 数据之间的粒度差异（Analyzing Data）
  - ✓ ***high-level, aggregated***与***fine-scaled, rich***的矛盾
- 通用接口与深度定制化（***Supporting Multiple Use Cases***）
  - ✓ 既需要通用性的接口来满足***casual usage***和***high-level scenarios***
  - ✓ 也需要深度定制化来准确定位问题

Example

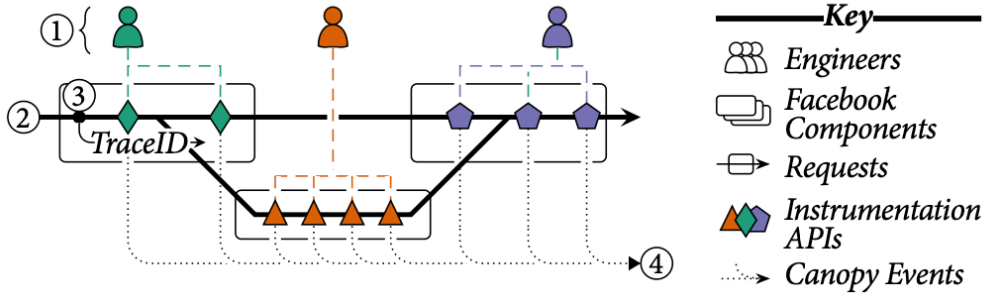

(a)
Page Load Latency
+13% Regression
Previous Week
+300ms
0
0.5
Time (days)
1.5
2
(b)
Critical Path Latency Breakdown
+50%
+25%
Prev Wk
−25%
Client
Browser
Resources
Network
Server
(c)
85%
75%
JS Prediction Accuracy
50%
40%
CSS Prediction Accuracy
−5%
(d)
2
1
0
Early-Flush Misses
+1
110kB
90kB
CSS Bytes
+10kB
(e)
Page Piece CSS Bytes
+10kB
Prev Wk
−5kB
Suggestions
MainContent
Navigation
UserInput
+10kB
(f)
Page Piece Critical Path Latency
+50%
+25%
Prev Wk
−25%
MainContent
UserInput
+150ms
Time (days)

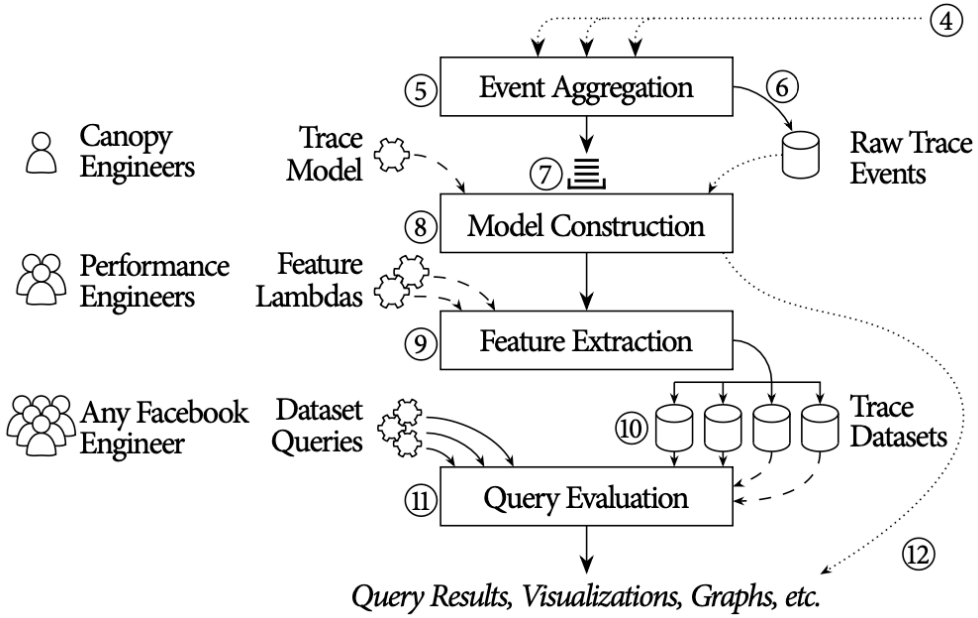

发现问题
(a)
关键路径分解
(b)
定位问题
(c)
进一步确认
(d)
根因分析
(e)
确认问题
(f)

Workflow

1. 工程师插桩
2. 请求进入被插桩组件
3. TraceID在组件中传播

4. 插桩函数捕获性能信息和因果关系
5. 性能事件被路由，执行聚合操作
6. 持久化存储
7. 以请求为粒度排队处理事件
8. 映射事件于Trace Model
9. 评估用户提供的Feature Lambdas
10. 数据集配置进行过滤和特征提取
11. 工程师直接查询可视化仪表盘
12. 深入检查





Instrumentation APIs

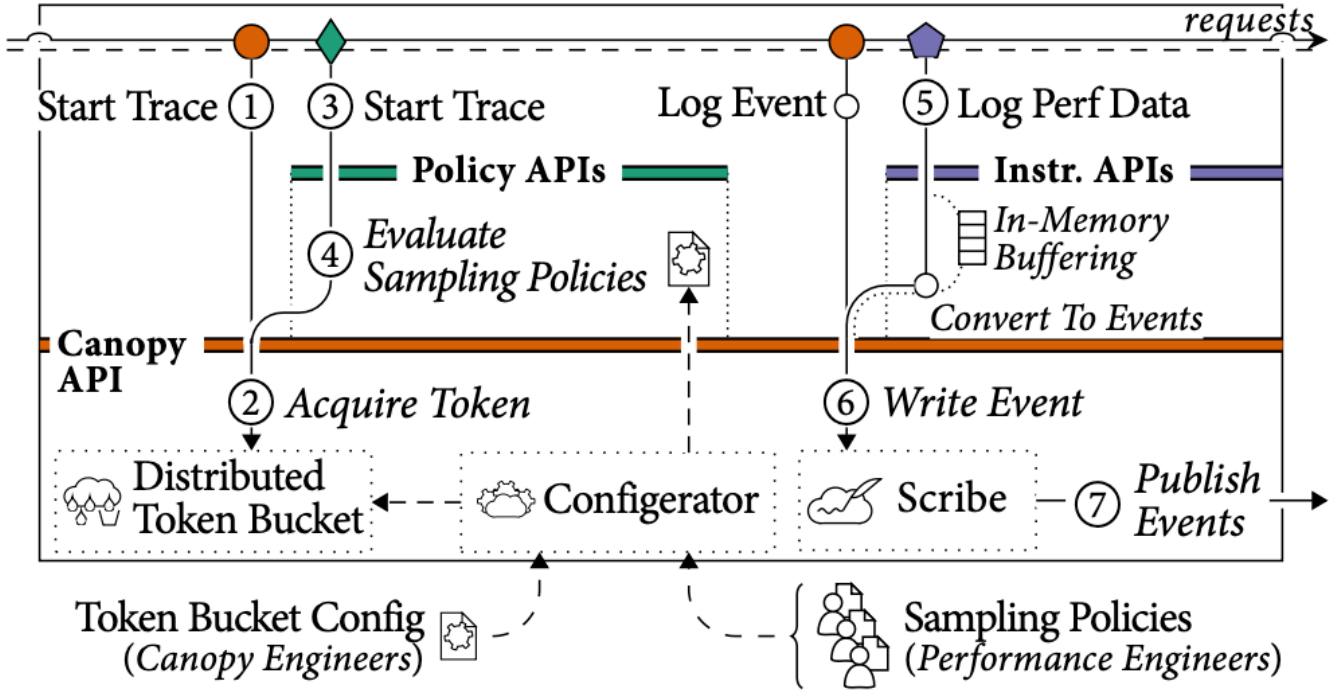
- 功能
  - ✓ 请求执行时传播***TraceID***
  - ✓ 记录请求结构
  - ✓ 捕获性能数据

```
try (Block b = Canopy.block("Doing some work")) { ... }
```

```
Canopy()->inform('Evicting Cache Entry');
Canopy()->measure('Evicting', $evictFunction);
```

Client Library

- Token Bucket
- Sampling Policies
- Opportunistic Tracing



Trace Events

- DAG
- 结构 TraceID
- Annotation 高层语义映射
- Sequence Number/Timestamp 统一进程/线程中事件顺序

```
struct Event {
  1: required string traceID;
  2: required string type;
  3: required string id1;
  4: optional string id2;
  5: optional i64 sequenceNumber;
  6: required i64 timestamp;
  7: optional map<string,string> annotations;
}
```

Figure 3: Thrift [55] declaration of Canopy events.

Modeled Traces

- ***Execution Units***：相当于一个执行线程
- ***Blocks***：一个***Execution Unit***内的执行片段
- ***Points***：一个***Block***内瞬时发生的事件
- ***Edges***：***Points***之间的非明显因果关系（***Enqueue***等）
  - ✓明显的因果关系已经包含在***Blocks***、***Points***等的结构中

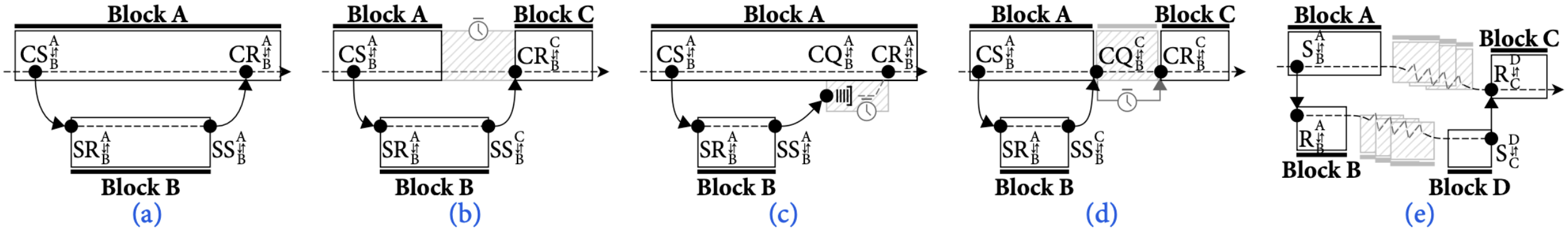


Figure 11: Evolution of Canopy's instrumentation and model for RPCs and network communication (cf. §5.3)

Pipeline

- **Routing**：1–3
  - ✓ 轮询
  - ✓ *Hbase+In-memory*
- **Handling**：4
  - ✓处理触发
- **Load Shedding**：5–7
  - ✓延时处理
- **Model Construction**
  - ✓形式验证（*Builder*）
- **Feature Extraction**：8–9
  - ✓公平队列

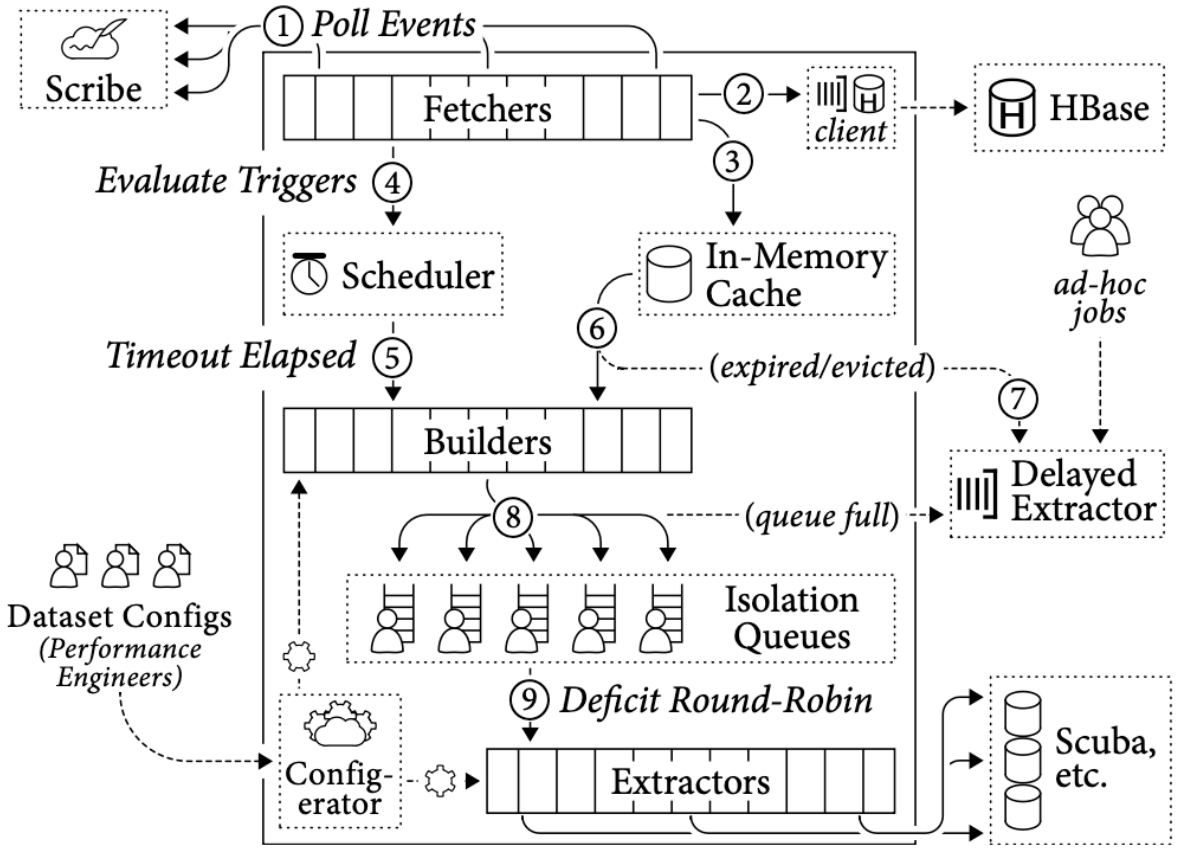


Figure 7: Tailer architecture; see §4.2 for description.

可扩展性与特征提取

- 可扩展性
  - ✓ ***Events*按*TraceID*分片处理**
  - ✓ 可指定跟踪事件粒度
  - ✓ ***pipeline*的在消息队列中的堆积**
- 特征提取
  - ✓ ***Domain-Specific Language***

(a) *Find the earliest client-side point*

```
Begin = ExecUnits | Filter(name="Client") | Points | First
```

(b) *Find the client-side "display done" marker.*

```
DisplayDone = ExecUnits | Filter(name="Client") | Points |
              Filter(marker="display_done") | First
```

(c) *Calculate display done latency.*

```
DisplayDone | Timestamp | Subtract(Begin | Timestamp) |
 RecordAs("display_done")
```

(d) *Find all network resource loading on the critical path.*

```
ResourceEdges = CriticalPath(Begin->DisplayDone) | Edges |
                Filter(type="resource")
```

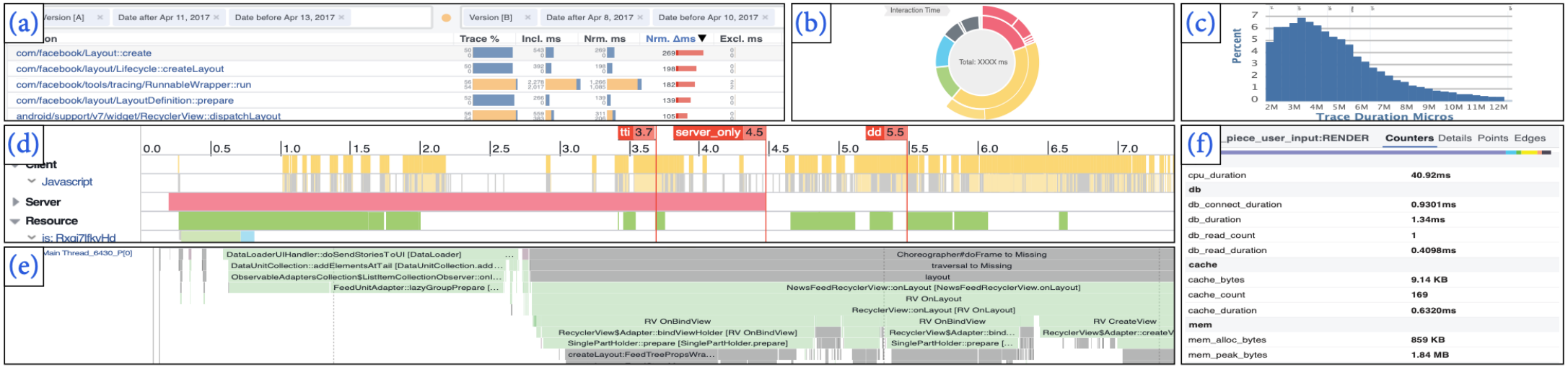
(e) *Calculate the total resource loading time on the critical path.*

```
ResourceEdges | Duration | Sum | RecordAs("res_load")
```

(f) *Calculate the total CSS bytes on the critical path.*

```
ResourceEdges | Filter(resource="css") | Counter(bytes) |
 Sum | RecordAs("css_bytes")
```

可视化

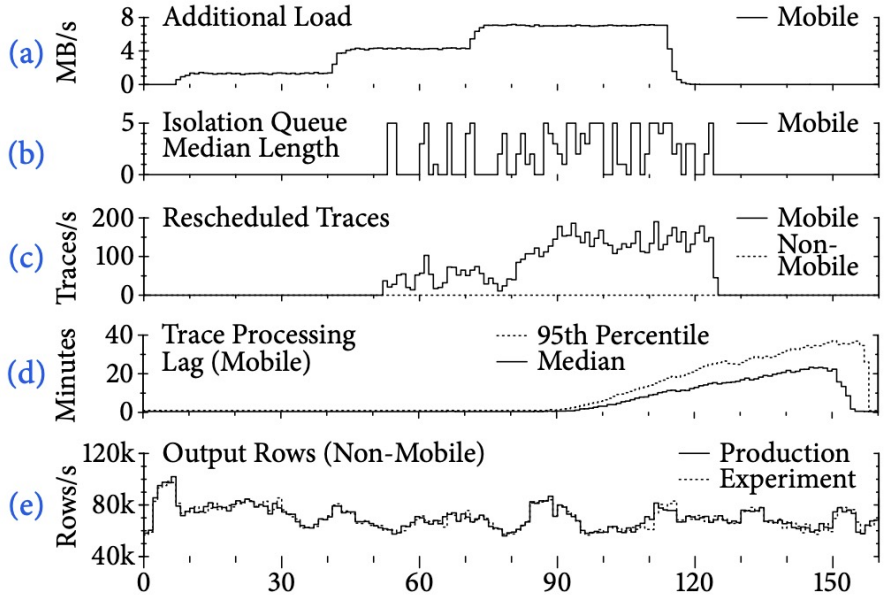

(a)
Version [A]
Date after Apr 11, 2017
Date before Apr 13, 2017
Version [B]
Date after Apr 8, 2017
Date before Apr 10, 2017
Trace %
Incl. ms
Nrm. ms
Nrm. Δms
Excl. ms
com/facebook/Layout::create
com/facebook/layout/Lifecycle::createLayout
com/facebook/tools/tracing/RunnableWrapper::run
com/facebook/layout/LayoutDefinition::prepare
android/support/v7/widget/RecyclerView::dispatchLayout
(b)
Interaction Time
Total: XXXX ms
(c)
Percent
Trace Duration Micros
(d)
tti 3.7
server_only 4.5
dd 5.5
Javascript
Server
Resource
(e)
Main Thread_6430_P[0]
DataLoaderUIHandler::doSendStoriesToUI [DataLoader]
DataUnitCollection::addElementsAtTail [DataUnitCollection.add...
ObservableAdaptersCollection$ListItemCollectionObserver::onI...
FeedUnitAdapter::lazyGroupPrepare [...
Choreographer#doFrame to Missing
traversal to Missing
layout
NewsFeedRecyclerView::onLayout [NewsFeedRecyclerView.onLayout]
RV OnLayout
RecyclerView::onLayout [RV OnLayout]
RV OnBindView
RV OnBindView
RV CreateView
RecyclerView$Adapter::bindViewHolder [RV OnBindView]
RecyclerView$Adapter::bind...
RecyclerView$Adapter::createV
SinglePartHolder::prepare [SinglePartHolder.prepare]
SinglePartHolder::prepare [...
(f)
_piece_user_input:RENDER
Counters
Details
Points
Edges
cpu_duration 40.92ms
db
db_connect_duration 0.9301ms
db_duration 1.34ms
db_read_count 1
db_read_duration 0.4098ms
cache
cache_bytes 9.14 KB
cache_count 169
cache_duration 0.6320ms
mem
mem_alloc_bytes 859 KB
mem_peak_bytes 1.84 MB

Evaluation

- ***Overhead***

		Mean	1st	25th	50th	75th	99th
WWW	CPU	6.76%	11.11%	7.31%	6.86%	6.44%	4.44%
	Wallclock	2.28%	7.11%	2.18%	2.14%	2.38%	2.70%
Android	Cold Start	8.57%	5.29%	4.88%	10.78%	13.37%	0.00%
Service	ServiceA Wallclock	8.15%	6.83%	7.03%	7.15%	7.53%	10.44%
	ServiceB Wallclock	0.76%	0.72%	0.79%	0.80%	0.79%	0.38%

- 公平性与处理滞后



On the Use of ML for Blackbox System Performance Prediction

NSDI 2021

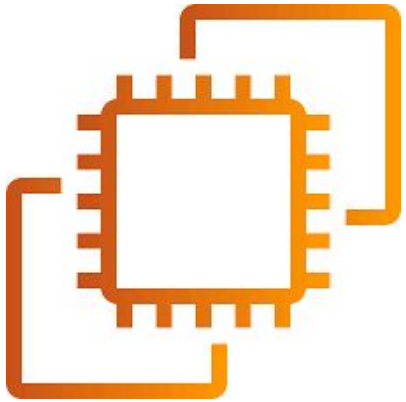
Silvery Fu, UC Berkeley; Saurabh Gupta, UIUC
Radhika Mittal, UIUC; Sylvia Ratnasamy, UC Berkeley

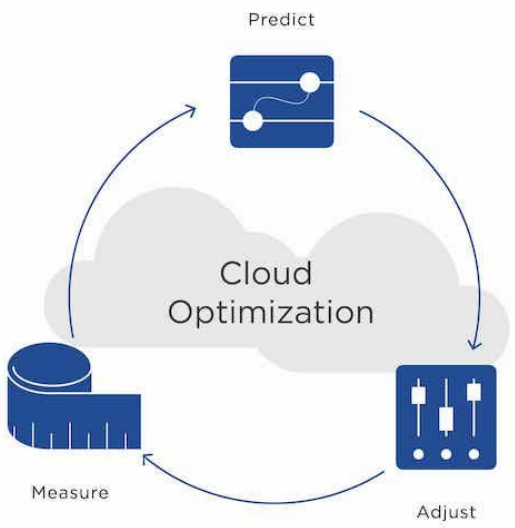
On the Use of ML for Blackbox System Performance Prediction

Silvery Fu[1], Saurabh Gupta[2], Radhika Mittal[2], Sylvia Ratnasamy[1]
(1: UC Berkeley, 2: UIUC)

Performance prediction is increasingly important!

- Optimization, capacity planning, SLO-aware scheduling





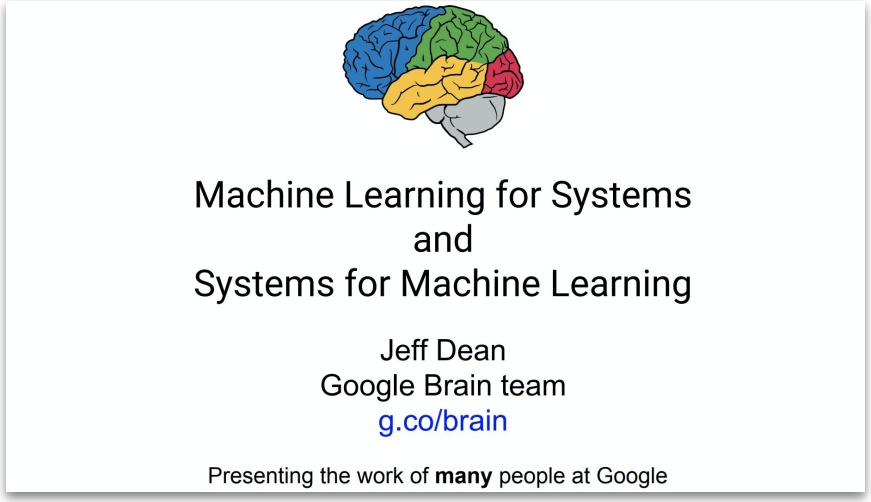
https://opsani.com/resources/cloud-optimization/

F(parameters) → performance

E.g., how many workers, size of input, machine configurations → JCT, query latency

Challenges

- Accurate
  - precise predictions
- Simple/easy-to-use
  - in-depth understanding of the systems not required
- General
  - works across a spectrum of workloads and applications



Can ML provide an *accurate*, *general*, and *simple* performance predictor?

ML for system perf. prediction?

This paper: a *systematic* and *broad* study on performance prediction!

Selecting the *Best* VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach

Neeraja J. Yadwadkar
University of California, Berkeley
neerajay@eecs.berkeley.edu

Bharath Hariharan
Facebook AI Research
bharathh@fb.com

Joseph E. Gonzalez
University of California, Berkeley
jegonzal@eecs.berkeley.edu

Burton Smith
Microsoft Research
burtons@microsoft.com

Randy H. Katz
University of California, Berkeley
randy@cs.berkeley.edu

ABSTRACT

Users of cloud services are presented with a bewildering choice of VM types and the choice of VM can have significant implications on performance and cost. In this paper we address the fundamental problem of *accurately* and *economically* choosing the *best VM* for a given *workload* and *user goals*. To address the problem of optimal VM selection, we present PARIS, a data-driven system that uses a novel *hybrid* offline and online data collection and modeling framework to provide *accurate* performance estimates with *minimal* data collection. PARIS is able to predict workload performance for different user-specified metrics, and resulting costs for a wide range of VM types and workloads across multiple cloud providers. When compared to sophisticated baselines, including collaborative filtering and a linear interpolation model using measured workload performance on two VM types, PARIS produces significantly better estimates of performance. For instance, it reduces runtime prediction error by a factor of 4 for some workloads on both AWS and Azure. The increased accuracy translates into a 45% reduction in user cost while maintaining performance.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **General and reference** → **Performance**; **Estimation**; • **Social and professional topics** → *Pricing and resource allocation*;

KEYWORDS

Cloud Computing, Resource Allocation, Performance Prediction, Data-Driven Modeling

ACM Reference Format:
Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. 2017. Selecting the *Best* VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach. In *Proceedings of SoCC '17, Santa Clara, CA, USA, September 24–27, 2017,* 14 pages.
https://doi.org/10.1145/3127479.3131614

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SoCC '17, September 24–27, 2017, Santa Clara, CA, USA

© 2017 Association for Computing Machinery.

ACM ISBN 978-1-4503-5028-0/17/09…$15.00
https://doi.org/10.1145/3127479.3131614

1 INTRODUCTION

As companies of all sizes migrate to cloud environments, increasingly diverse workloads are being run in the Cloud – each with different performance requirements and cost trade-offs [59]. Recognizing this diversity, cloud providers offer a wide range of Virtual Machine (VM) types. For instance, at the time of writing, Amazon [2], Google [7], and Azure [42] offered a combined total of over 100 instance types with varying system and network configurations.

In this paper we address the fundamental problem of *accurately* and *economically* choosing the *best VM* for a given *workload* and *user goals*. This choice is critical because of its impact on performance metrics such as runtime, latency, throughput, cost, and availability. Yet determining or even defining the *"best"* VM depends heavily on the users' goals which may involve diverse, application-specific performance metrics, and span tradeoffs between price and performance objectives.

For example, Figure 1 plots the runtimes and resulting costs of running a video encoding task on several AWS VM types. A typical user wanting to deploy a workload might choose the cheapest VM type (`m1.large`) and paradoxically end up not just with poor performance but also high total costs. Alternatively, overprovisioning by picking the most expensive VM type (`m2.4xlarge`) might only offer marginally better runtimes than much cheaper alternatives like `c3.2xlarge`. Thus, to choose the right VM for her performance goals and budget, the user needs accurate performance estimates.

Recent attempts to help users select VM types have either focused on optimization techniques to efficiently search for the best performing VM type [12], or extensive experimental evaluation to model the performance cost trade-off [69]. Simply optimizing for the best VM type for a particular goal (as in CherryPick [12]) assumes that this goal is *fixed*; however, different users might prefer different points along the performance-cost trade-off curve. For example, a user might be willing to tolerate mild reductions in performance for substantial cost savings. In such cases, the user might want to know precisely how switching to another VM type affects performance and cost.

The alternative, directly modeling the performance-cost tradeoff, can be challenging. The published VM characteristics (e.g., memory and virtual cores) have hard-to-predict performance implications for any given workload [24, 39, 72]. Furthermore, the performance often depends on workload characteristics that are difficult to specify [15, 28, 39]. Finally, variability in the choice of host hardware, placement policies, and resource contention [59] can result in performance variability [29, 35, 54] that is not captured

Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics

Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, Ion Stoica
University of California, Berkeley

Abstract

Recent workload trends indicate rapid growth in the deployment of machine learning, genomics and scientific workloads on cloud computing infrastructure. However, efficiently running these applications on shared infrastructure is challenging and we find that choosing the right hardware configuration can significantly improve performance and cost. The key to address the above challenge is having the ability to predict performance of applications under various resource configurations so that we can automatically choose the optimal configuration.

Our insight is that a number of jobs have predictable structure in terms of computation and communication. Thus we can build performance models based on the behavior of the job on small samples of data and then predict its performance on larger datasets and cluster sizes. To minimize the time and resources spent in building a model, we use optimal experiment design, a statistical technique that allows us to collect as few training points as required. We have built Ernest, a performance prediction framework for large scale analytics and our evaluation on Amazon EC2 using several workloads shows that our prediction error is low while having a training overhead of less than 5% for long-running jobs.

1 Introduction

In the past decade we have seen a rapid growth of large-scale *advanced* analytics that implement complex algorithms in areas like distributed natural language processing [24, 74], deep learning for image recognition [34], genome analysis [72, 61], astronomy [17] and particle accelerator data processing [19]. These applications differ from traditional analytics workloads (e.g., SQL queries) in that they are not only data-intensive but also computation-intensive, and typically run for a long time (and hence are expensive). Along with new workloads, we have seen widespread adoption of cloud computing with large data sets being hosted [7, 1], and the emergence of sophisticated analytics services, such as machine learning, being offered by cloud providers [9, 6].

With cloud computing environments such as Amazon EC2, users typically have a large number of choices in terms of the instance types and number of instances they can run their jobs on. Not surprisingly, the amount of memory per core, storage media, and the number of instances are crucial choices that determine the running time and thus indirectly the cost of running a given job. Using common machine learning kernels we show in §2.2 that choosing the right configuration can improve performance by up to 1.9x at the same cost.

In this paper, we address the challenge of choosing the configuration to run large advanced analytics applications in heterogeneous multi-tenant environments. The choice of configuration depends on the user's goals which typically includes either minimizing the running time given a budget or meeting a deadline while minimizing the cost. The key to address this challenge is developing a performance prediction framework that can accurately predict the running time on a specified hardware configuration, given a job and its input.

One approach to address this challenge is to predict the performance of a job based on monitoring the job's previous runs [39, 44]. While simple, this approach assumes the job runs repeatedly on the same or "similar" data sets. However, this assumption does not always hold. First, even when a job runs periodically it typically runs on data sets that can be widely different in both size and content. For example, a prediction algorithm may run on data sets corresponding to different days or time granularities. Second, workloads such as interactive machine learning [9, 55] and parameter tuning generate unique jobs for which we have little or no relevant history. Another approach to predict job performance is to build a detailed parametric model for the job. Along these lines, several techniques have been recently proposed in the context of MapReduce-like frameworks [77, 52]. These techniques have been aided by the inherent simplicity of the two-stage MapReduce model. However, the recent increase in the popularity of more complex parallel computation engines such as Dryad [51] and Spark [83] make these parametric techniques much more difficult to apply.

In this paper, we propose a new approach that can accurately predict the performance of a given analytics job. The main idea is to run a set of instances of the entire job on samples of the input, and use the data from these training runs to create a performance model. This approach has low overhead, as in general it takes much less time and resources to run the training jobs than running the job itself. Despite the fact that this is a black-box approach (i.e., requires no knowledge about the internals of

Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics

Ana Klimovic — *Stanford University*
Heiner Litz — *UC Santa Cruz*
Christos Kozyrakis — *Stanford University*

Abstract

Data analytics are an important class of data-intensive workloads on public cloud services. However, selecting the right compute and storage configuration for these applications is difficult as the space of available options is large and the interactions between options are complex. Moreover, the different data streams accessed by analytics workloads have distinct characteristics that may be better served by different types of storage devices.

We present Selecta, a tool that recommends near-optimal configurations of cloud compute and storage resources for data analytics workloads. Selecta uses latent factor collaborative filtering to predict how an application will perform across different configurations, based on sparse data collected by profiling training workloads. We evaluate Selecta with over one hundred Spark SQL and ML applications, showing that Selecta chooses a near-optimal performance configuration (within 10% of optimal) with 94% probability and a near-optimal cost configuration with 80% probability. We also use Selecta to draw significant insights about cloud storage systems, including the performance-cost efficiency of NVMe Flash devices, the need for cloud storage with support for fine-grain capacity and bandwidth allocation, and the motivation for end-to-end storage optimizations.

1 Introduction

The public cloud market is experiencing unprecedented growth, as companies move their workloads onto platforms such as Amazon AWS, Google Cloud Platform and Microsoft Azure. In addition to offering high elasticity, public clouds promise to reduce the total cost of ownership as resources can be shared among tenants. However, achieving performance and cost efficiency requires choosing a suitable configuration for each given application. Unfortunately, the large number of instance types and configuration options available make selecting the right resources for an application difficult.

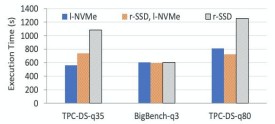


Figure 1: Performance of three applications on eight `i3.xl` instances with different storage configurations.

The choice of storage is often essential, particularly for cloud deployments of data-intensive analytics. Cloud vendors offer a wide variety of storage options including object, file and block storage. Block storage can consist of hard disks (*HDD*), solid-state drives (*SSD*), or high bandwidth, low-latency NVMe Flash devices (*NVMe*). The devices may be local (*l*) to the cloud instances running the application or remote (*r*). These options alone lead to storage configuration options that can differ by orders of magnitude in terms of throughput, latency, and cost per bit. The cloud storage landscape is only becoming more diverse as emerging technologies based on 3D X-point become available [35, 16].

Selecting the right cloud storage configuration is critical for both performance and cost. Consider the example of a Spark SQL equijoin query on two 128 GB tables [53]. We find the query takes 8.7× longer when instances in an 8-node EC2 cluster access *r*-HDD compared to *l*-NVMe storage. This is in contrast to a recent study, conducted with a prior version of Spark, which found that faster storage can only improve the median job execution time by at most 19% [50]. The performance benefits of *l*-NVMe lead to 8× lower execution cost for this query, even though NVMe storage has higher cost per unit time. If we also consider a few options for the number of cores and memory per instance, the performance gap between the best and worst performing VM-storage configurations is over 30×.

ML for system perf. prediction?

Start with the *best-case* scenario!

The **Best-Case** (BC) Test

- Given parameters P_1, P_2, P_3, …, P_k, want to learn F(P) → Perf. (e.g. JCT)
  - Dataset: data points of <P=X, JCT=Y>; split into training and testing sets

- ML assumptions:
  - *One-feature-at-a-time:* e.g., vary P_2, keeping P_1, P_3, … P_k fixed
  - *Seen-configuration:* e.g., points where P_2=1GB appear in training and testing-sets
- Systems assumptions:
  - *No-contention:* dedicated EC2 instances, isolated experiments;
  - *Identical-inputs:* same input data for a given input dataset size;

Applications and Models

Framework	Application/Description	Input Workload	Input Parameter	App. Config. Parameter	Infra. Parameter	Metric
Memcached [12]	Disributed in-memory k-v store	Mutilate [13]	value size	# servers	inst. type	mean query lat.
Nginx [9]	Web server, LB, Reverse Proxy	Wrk2 [5]	req. rate	# servers	inst. type	median req. lat.
Influxdb [15]	Open source time series database	Inch [10]	# points per timeseries	# servers	inst. type	mean query lat.
Go-fasthttp [7]	Fast HTTP package for Go	wrk2 [5]	# conn.	# servers	inst. type	median req. lat.
Spark [3]	*TeraSort:* sorting records	TeraGen	# records	# executors	inst. type	JCT
	PageRank: graph computation	GraphX SynthBenchmark [8]	# vertices			
	LR1: logistic regression	MLLib examples	# examples			
	LR2: logistic regression	Databricks Perf Test [16]				
	KMeans: clustering					
	Word2vec: feature extraction					
	FPGrowth: data mining					
	ALS: recommendation					
TensorFlow [17], Kubernetes [11]	*TFS:* Tensorflow model serving	Resnet examples [18]	# conn.	# servers	inst. type	requests/sec

ML models:

Nearest-neighbors,
Linear-regression,
Random forest,
SVM, SVM-kernelized,
Neural networks

Metrics and Predictors

- Accuracy metric:
  - rMSRE

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - f(X_i)}{Y_i}\right)^2}$$

- ML predictors → Best-of-Model/BoM-err
  - rMSRE of the most accurate model

- Oracle predictor → **O-err**

$$f_{\text{oracle}}(X) = \left(\sum_{i=1}^{n}\frac{\delta(X_i,X)}{Y_i}\right) / \left(\sum_{i=1}^{n}\frac{\delta(X_i,X)}{Y_i^2}\right),$$

$$\delta(a,b) = 1 \text{ if } a \text{ is equal to } b, \text{ and } 0 \text{ otherwise.}$$

Y_i: true value

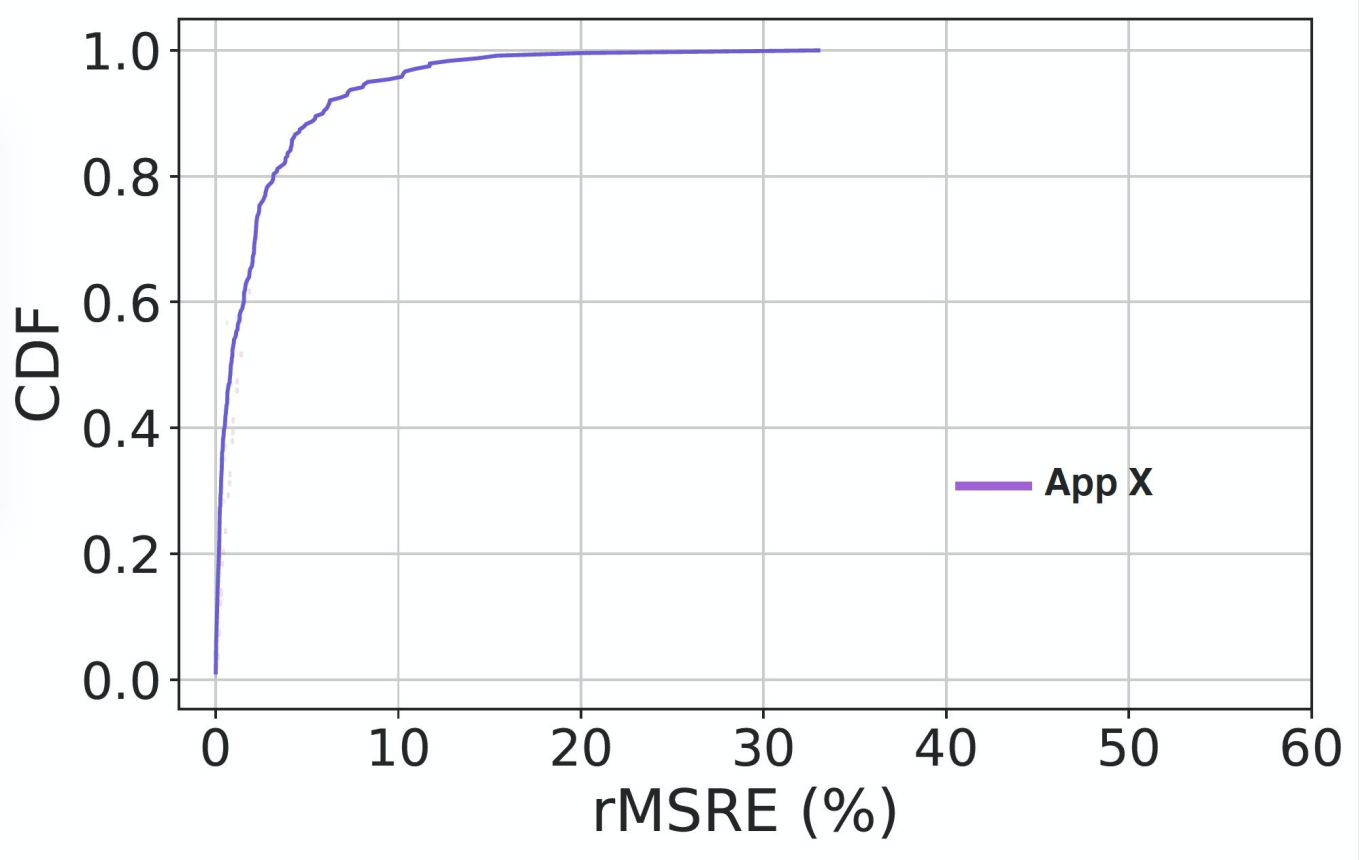
$f(X_i)$: predicted value

To obtain O-err:

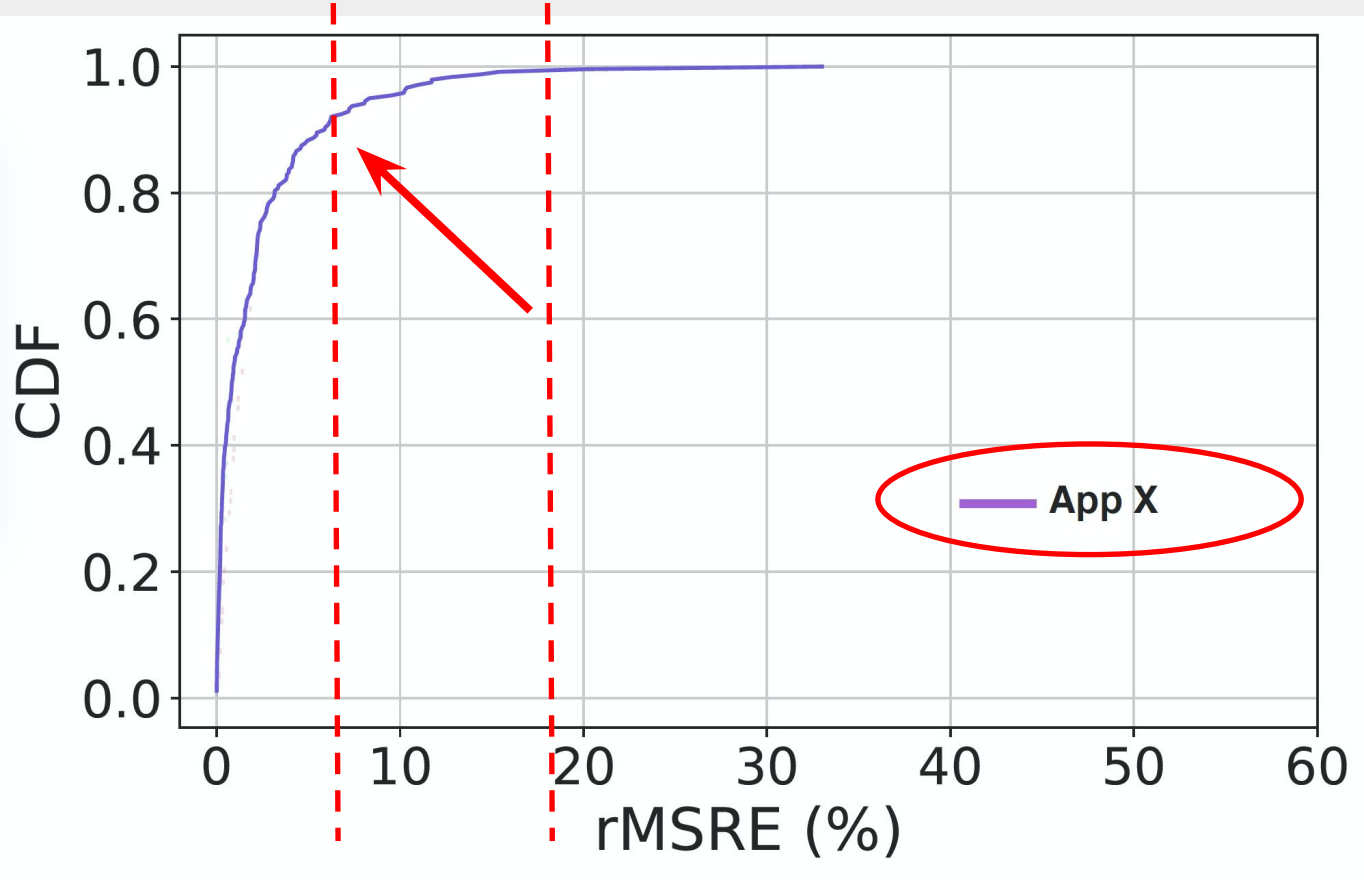
- Allow Oracle to peek at both the error function and test data!

- BoM-err ≥ O-err

Best Case Test Results



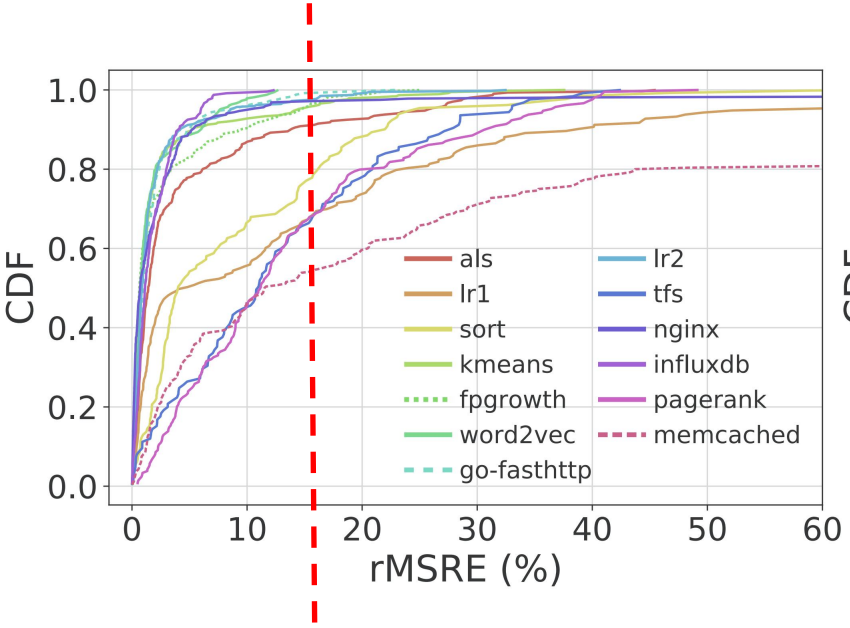
Best Case Test Results



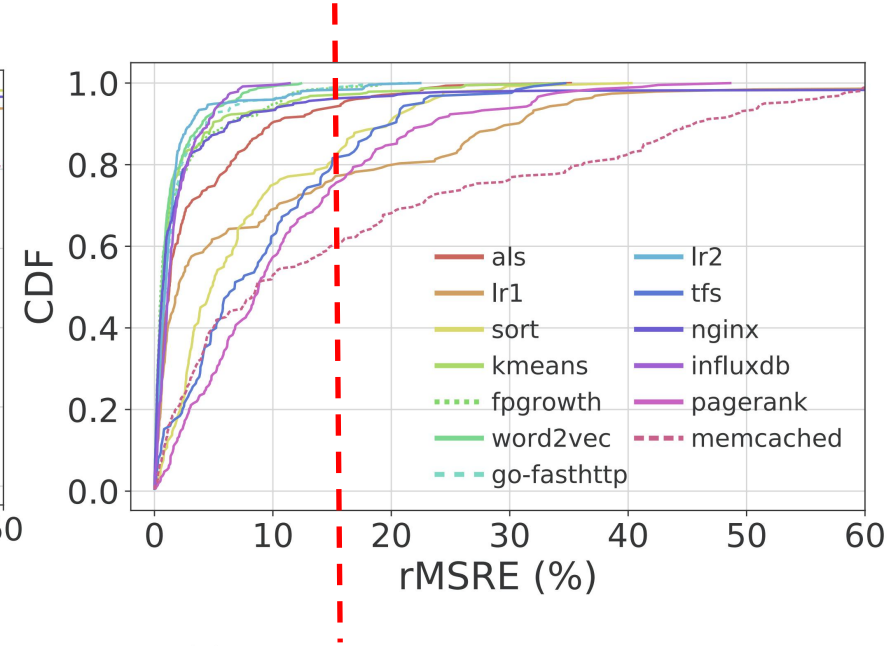
Error < 5% for 90% of predictions!

Error < 15% for ~99% predictions!

Best Case Test Results



(b) CDF of BoM-err in the BC test



(a) CDF of O-err in the BC test

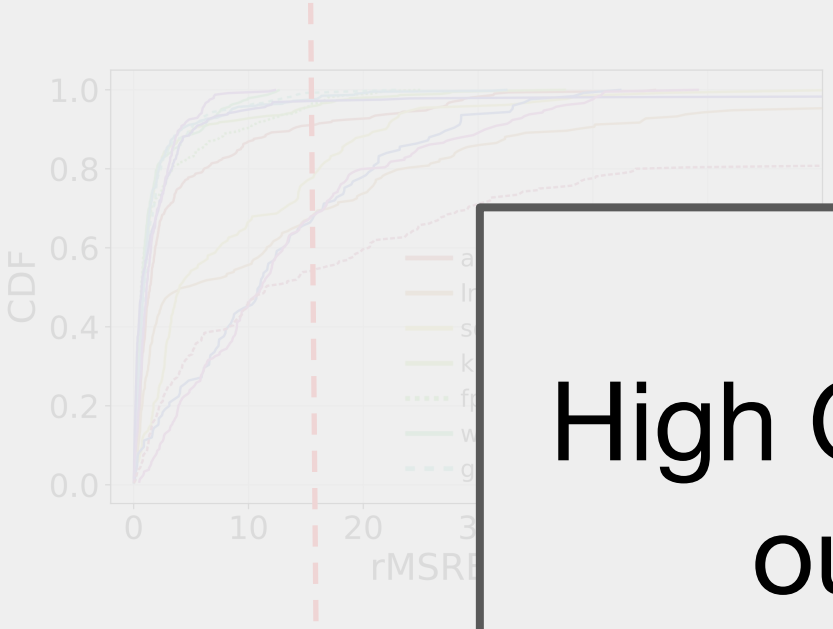
BoM-err: rMSRE from the most accurate model

O-err: rMSRE from the Oracle

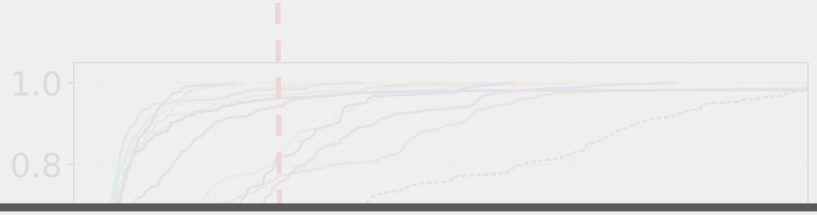
Observations:

- Despite best-case assumptions, the BoM often fails to achieve high accuracy.
- Oracle errors (the lower bound) are high.

Best Case Test Results



(b) CDF of BoM-err



SRE from the
e model

E from the Oracle

High Oracle error even under our best-case setup!

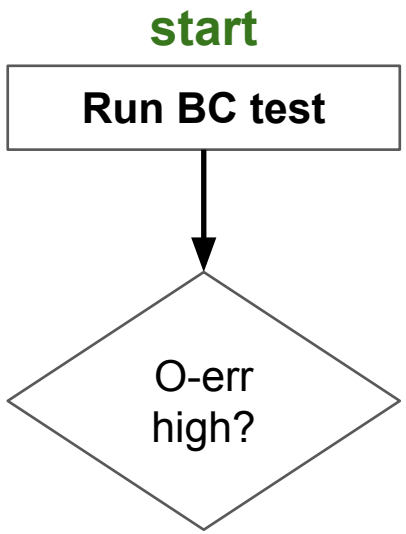
(Accurate ❌)

Observations:

- Despite bes ccuracy.
- Oracle errors (the lower bound) are high.

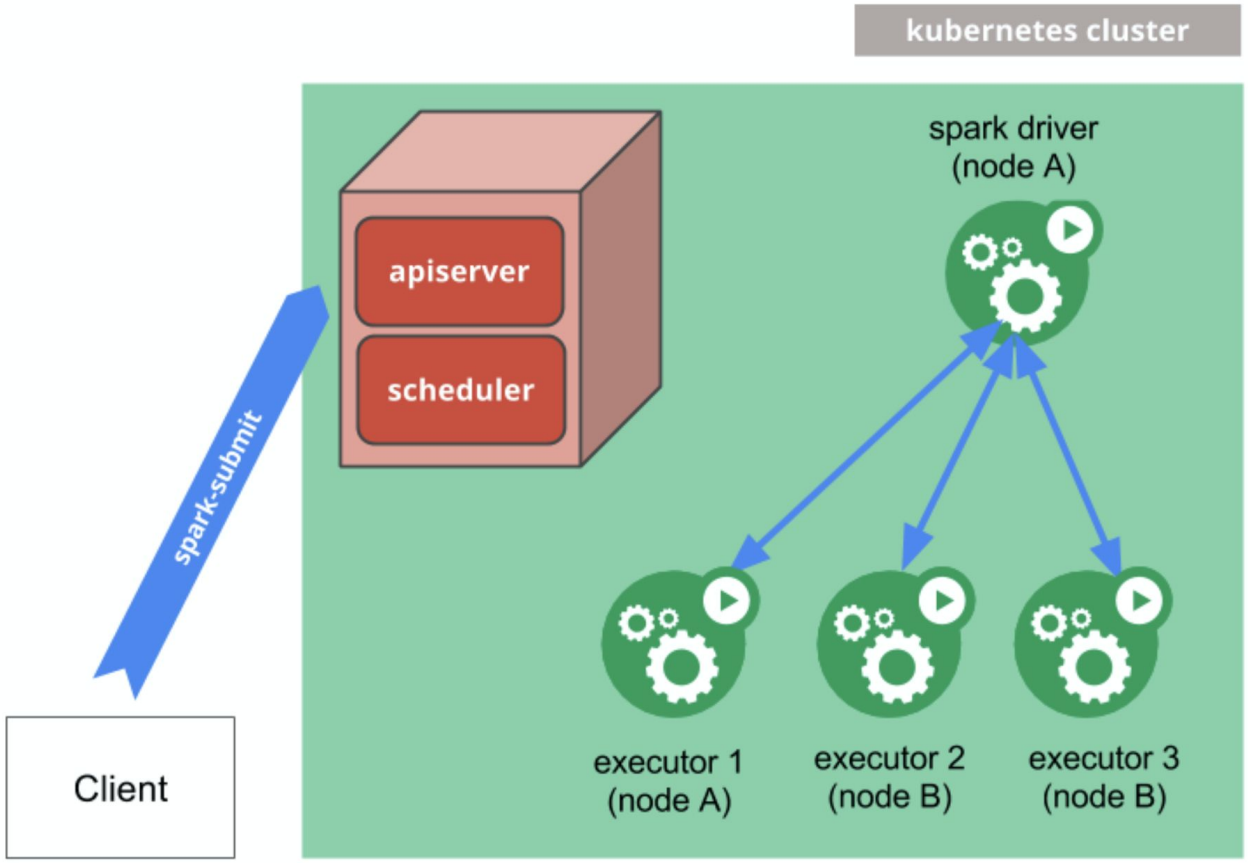
Methodology

Methodology



Root Cause	Applications Impacted
Spark's "start when 80% of workers are ready" optimization	Terasort
Multi-mode optimization in JVM Garbage Collector	LR1
Non-determinism in Spark sched.	PageRank
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS
Imperfect load-balancing at high load	TensorFlow serving
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx

E.g., Spark worker readiness



https://spark.apache.org/docs/latest/running-on-kubernetes.html

- Spark launches a job once at least 80% of target workers are ready

Root-causes

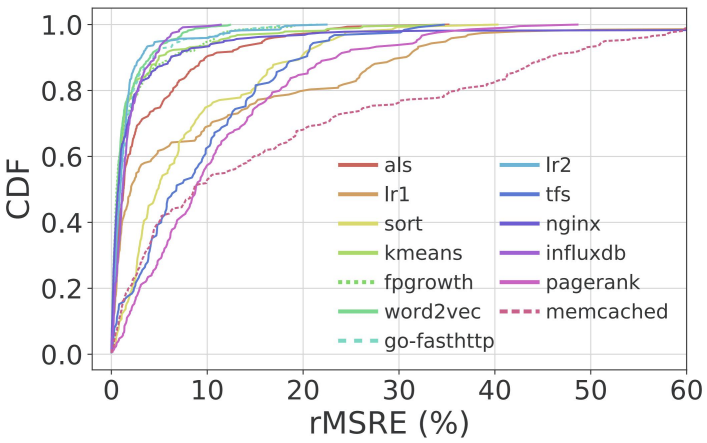
Fix?

Root Cause	Applications Impacted
Spark's "start when 80% of workers are ready" optimization	Terasort
Multi-mode optimization in JVM Garbage Collector	LR1
Non-determinism in Spark sched.	PageRank
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS
Imperfect load-balancing at high load	TensorFlow serving
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx

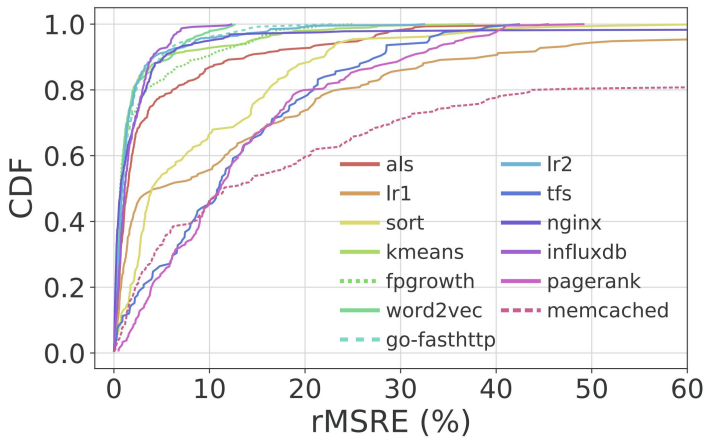
Root-causes Fix?

Root Cause	Applications Impacted	Modification
Spark's "start when 80% of workers are ready" optimization	Terasort	Disable optimization
Multi-mode optimization in JVM Garbage Collector	LR1	Avoid triggering, or disable, optimization
Non-determinism in Spark sched.	PageRank	Use deterministic data structure
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS	Client-side caching of HTTP redirects (OR always redirect)
Imperfect load-balancing at high load	TensorFlow serving	Modified load-balancing policy to always favor local workers
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx	Use AWS placement APIs / include inter-node RTTs as ML feature

With system modifications

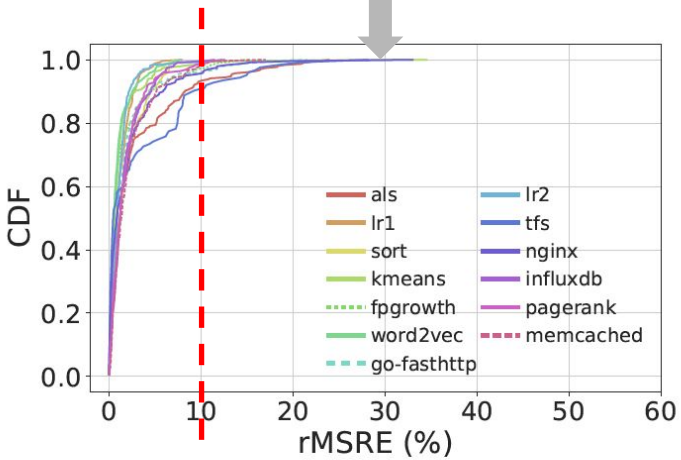


(a) CDF of O-err in the BC test

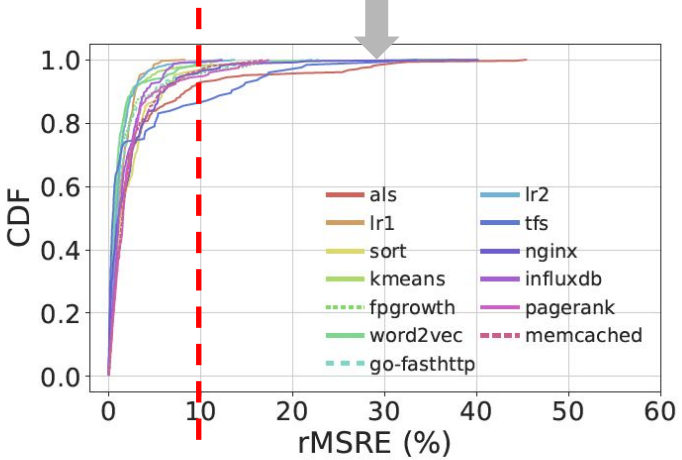


(b) CDF of BoM-err in the BC test

Before



(a) CDF of O-err in the BC test



(b) CDF of BoM-err in the BC test

After

- For all applications, Oracle error is now well within 10%!
- Best-of-Model error likewise!

All root-causes Fixes

Root Cause	Applications Impacted	Modification	Trade-off
Spark's "start when 80% of workers are ready" optimization	Terasort	Disable optimization	Decreased resilience to stragglers and worker failure
Multi-mode optimization in JVM Garbage Collector	LR1	Avoid triggering, or disable, optimization	Slower garbage collection
Non-determinism in Spark sched.	PageRank	Use deterministic data structure	None
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS	Client-side caching of HTTP redirects (OR always redirect)	Decreased flexibility[6] (OR slower name resolutions)
Imperfect load-balancing at high load	TensorFlow serving	Modified load-balancing policy to always favor local workers	Load imbalance when each server has different numbers of workers
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx	Use AWS placement APIs / include inter-node RTTs as ML feature	Cloud APIs expose more information (less flexibility)

- Trade-off between predictability and other design goals!
- E.g., disabling an optimization can lead to higher prediction accuracy but degraded performance

All root-causes

Fixes

Root Cause	Applications Impacted	Modification	Trade-off
Spark's "start when 8 are ready" opti			resilience to stragglers worker failure
Multi-mode optimiz Garbage Col			garbage collection
Non-determinism in			None
HTTP redirects and D S3's name res			flexibility[6] (OR slower me resolutions)
Imperfect load-bala load			ance when each server nt numbers of workers
Variability in imple Cloud APIs			APIs expose more tion (less flexibility)

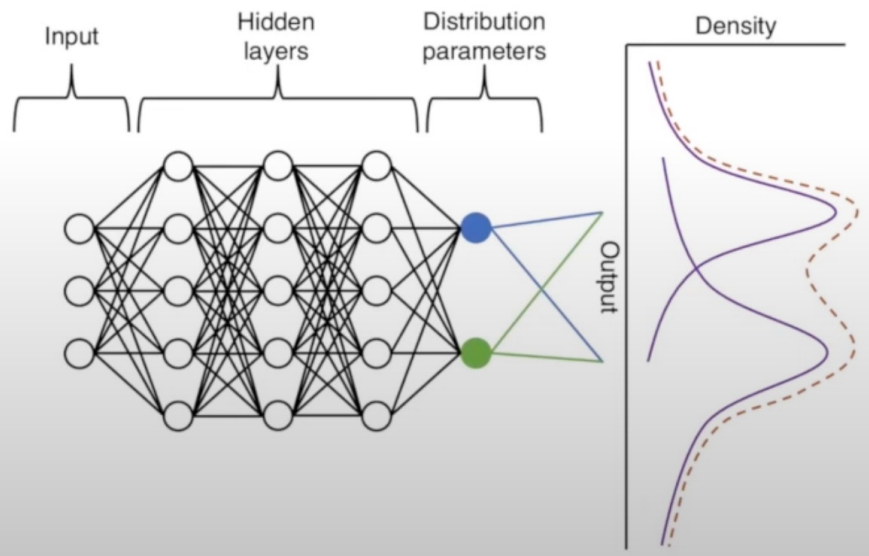
These "fixes" require in-depth understanding of the app. and reasoning about trade-offs!

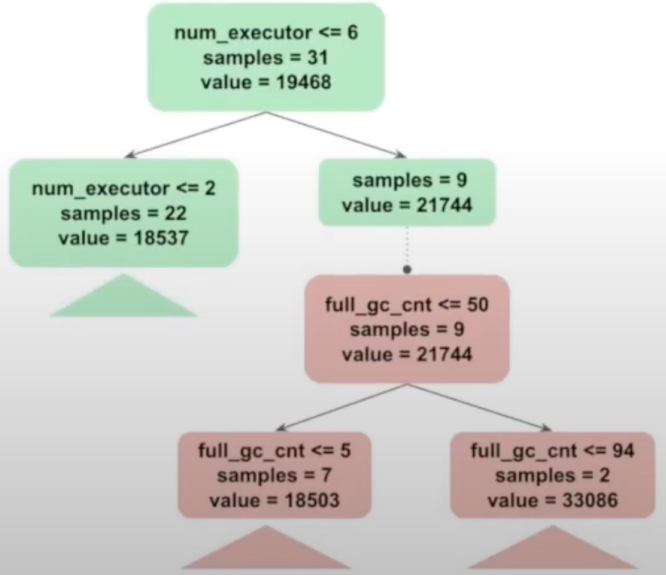
(Easy-to-use ❌)

- Trade-off b
- E.g., disabling an optimization can lead to higher prediction accuracy but degraded performance

Embrace variability: probabilistic predictions

- Idea: predicting a mixture distribution instead of a single value;
- Then, use the "modes" of each distribution as the "top-k" prediction value
  - E.g., identify the worst predicted performance that violates SLA
- ML: Mixture Density Networks and Probabilistic Random Forest





更多视频

doi: https://doi.org/10.1371/journal.pcbi.1006869.g001

Embrace variability: probabilistic predictions

- Idea: predicting a mixture distribution instead of a single value;
- Then, use the "modes" of each distribution as the "top-k" prediction value
  - E.g., id
- ML: Mixtu

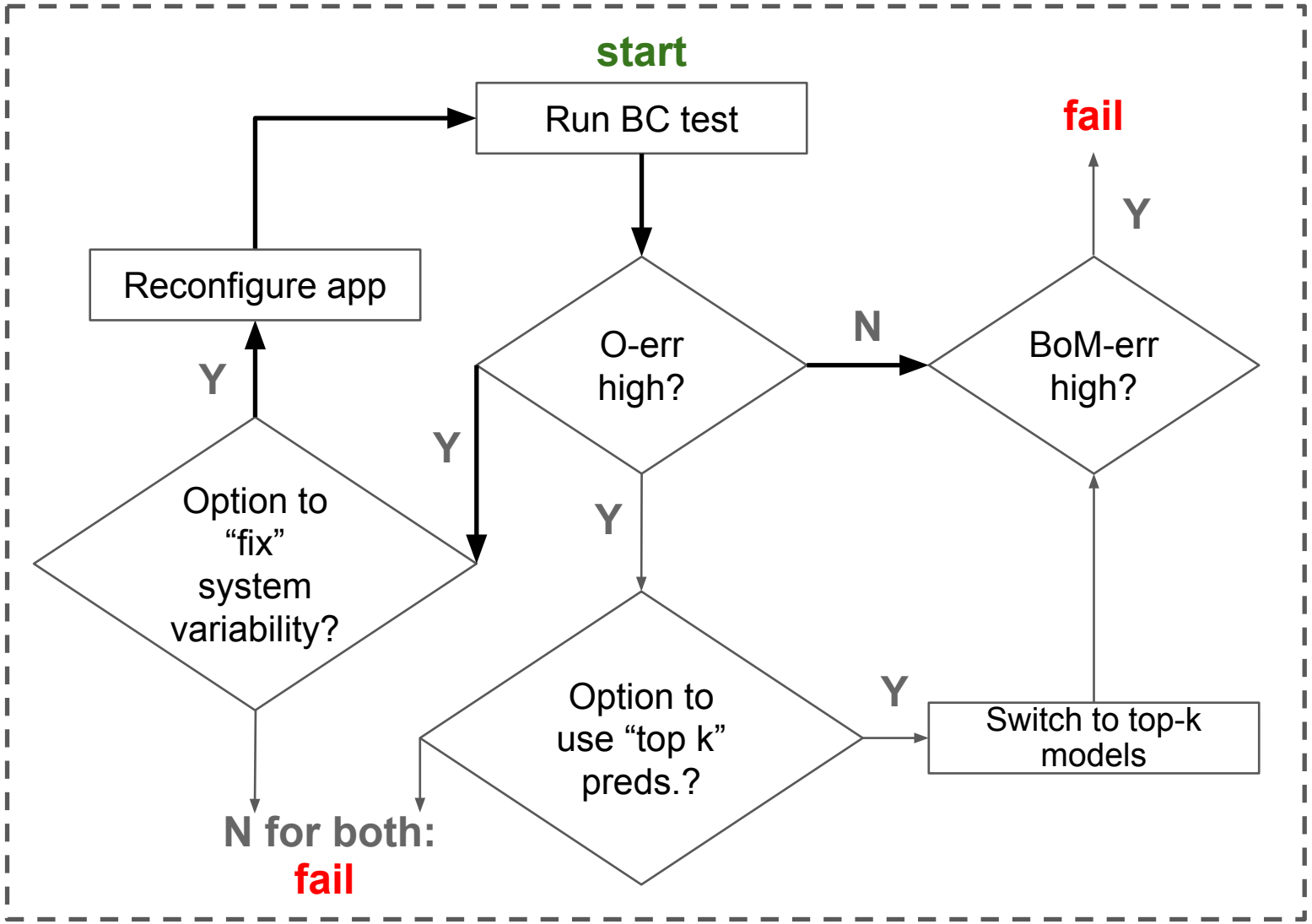
Significant decrease in BoM-err with top-3 (k=3) predictions!

But top-k predictions may not be useful to all cases!

(General ❌)

doi: https://doi.org/10.1371/journal.pcbi.1006869.g001

Methodology



So far, *best-case* setup only!

- one-feature-at-a-time
- seen-configuration
- no-contention
- identical-inputs

What if we go "beyond the best case"?

- Relaxing the **one-feature-at-a-time** assumption:
  - vary all parameters!
- Relaxing the **seen-configuration** assumption:
  - configuration-to-predict is never seen during model training!
- Relaxing the **no-contention** assumption:
  - use default/shared EC2 instances!
- Relaxing the **identical-inputs** assumption:
  - varied datasets (e.g., different random seeds in data generation)

Run on modified systems with the fixes!

What if we go "beyond the best case"?

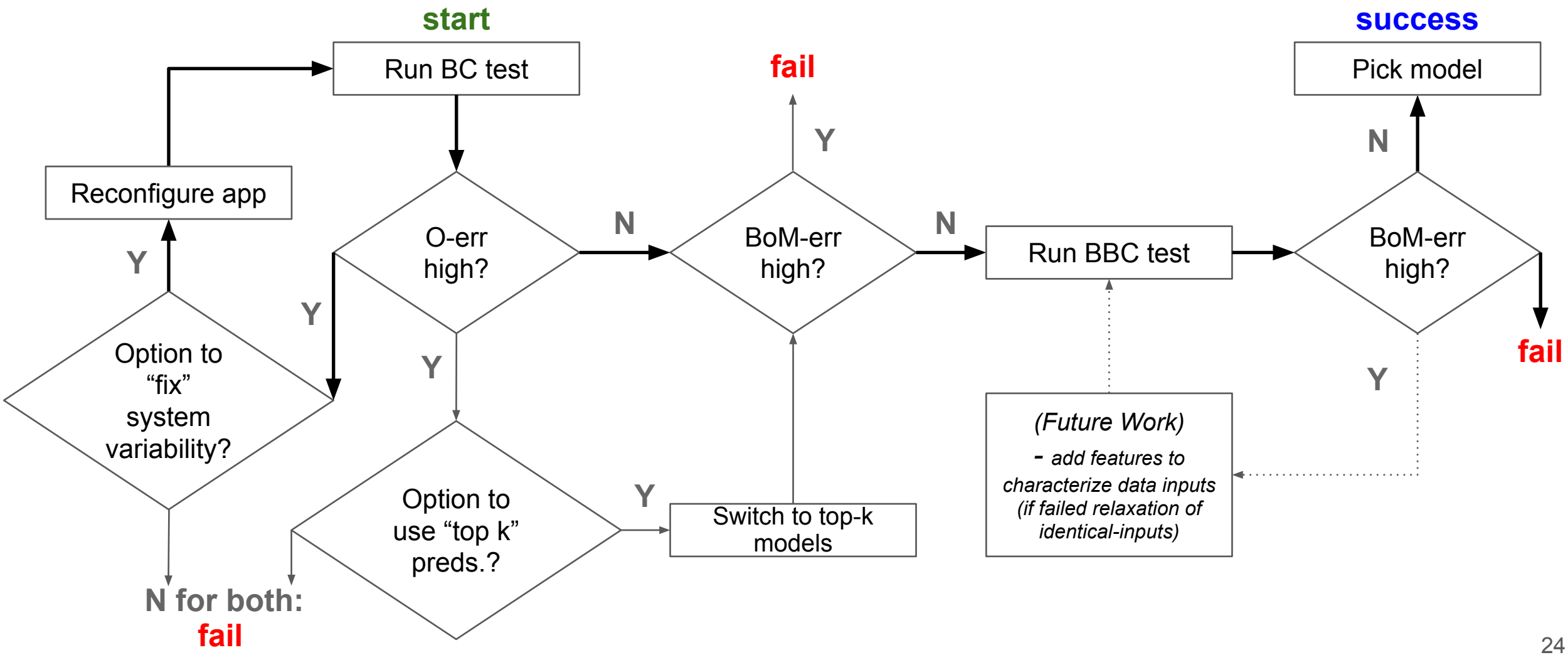
- Relaxing the **one-feature-at-a-time** assumption:
  - vary all parameters!
- Relaxing the
  - configura
- Relaxing the
  - use defau
- Relaxing the
  - varied datasets (e.g., different random seeds in data generation)

on modified
ems with the
fixes!

Prediction errors can remain high if the underlying performance trend is difficult to learn!

(General ❌)

Methodology Blueprint



Conclusion:

- Taken "out of the box", many apps exhibit a surprisingly *high degree of irreducible error*

- We *can* significantly improve the accuracy if we accept the loss of simplicity and/or generality:
  - modify applications
  - modify predictions
  - ..but they don't work in all cases

- Need a more nuanced methodology for applying ML

Conclusion:

- Accurate
  - precise predictions
- Simple/easy-to-use
  - in-depth understanding of the systems not required
- General
  - works across a spectrum of workloads and applications

Can ML provide an accurate, general, and simple performance predictor?

No.

Thanks!