Google's Dapper – A Large-Scale Distributed Systems Tracing Infrastructure

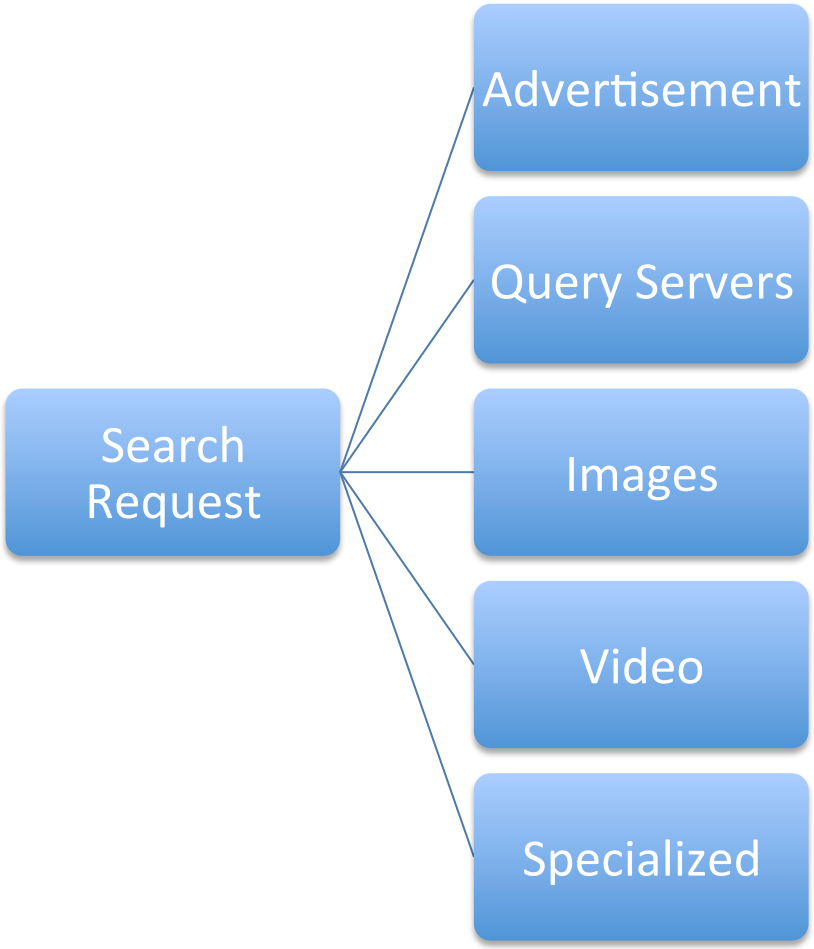
Prasanna Gautam

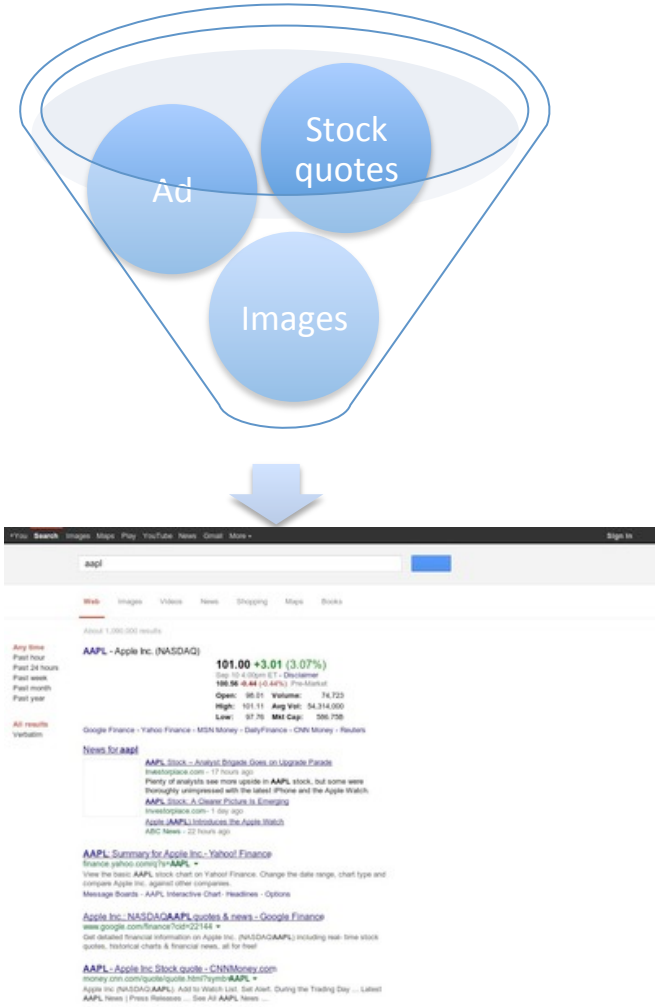
9/11/2014 (UConn CSE5095-005)

Goals

- Provide developers with
  - Information about behavior of complex distributed systems, applications
  - Tools to debug and analyze production service issues
  - Application level transparency and service dependencies
  - Assist in minimizing end to end latencies

Example: Web Search





Example: Web Search

- Many services
  - A lot of people writing, updating them constantly
- Performance sensitive
  - End to End latency for search needs to be low
- On shared hardware with other services competing for resources

Potential solution

- Write a library or framework that is
  - Embedded into majority of services
  - Continuously collecting and monitoring data
  - Quick turnaround time

Dapper: Requirements

- Low-Overhead
  - Cannot affect performance significantly
- Application Level Transparency
  - Who did what, when, how(RPC?)
- Scalability
  - At Google Scale, can't trace everything.. Sample!
- Making tracing data available quickly (< 1 min)

Dapper: Requirements

- Not an experimental, fringe system
- Runs on production hardware and services
- Tracing

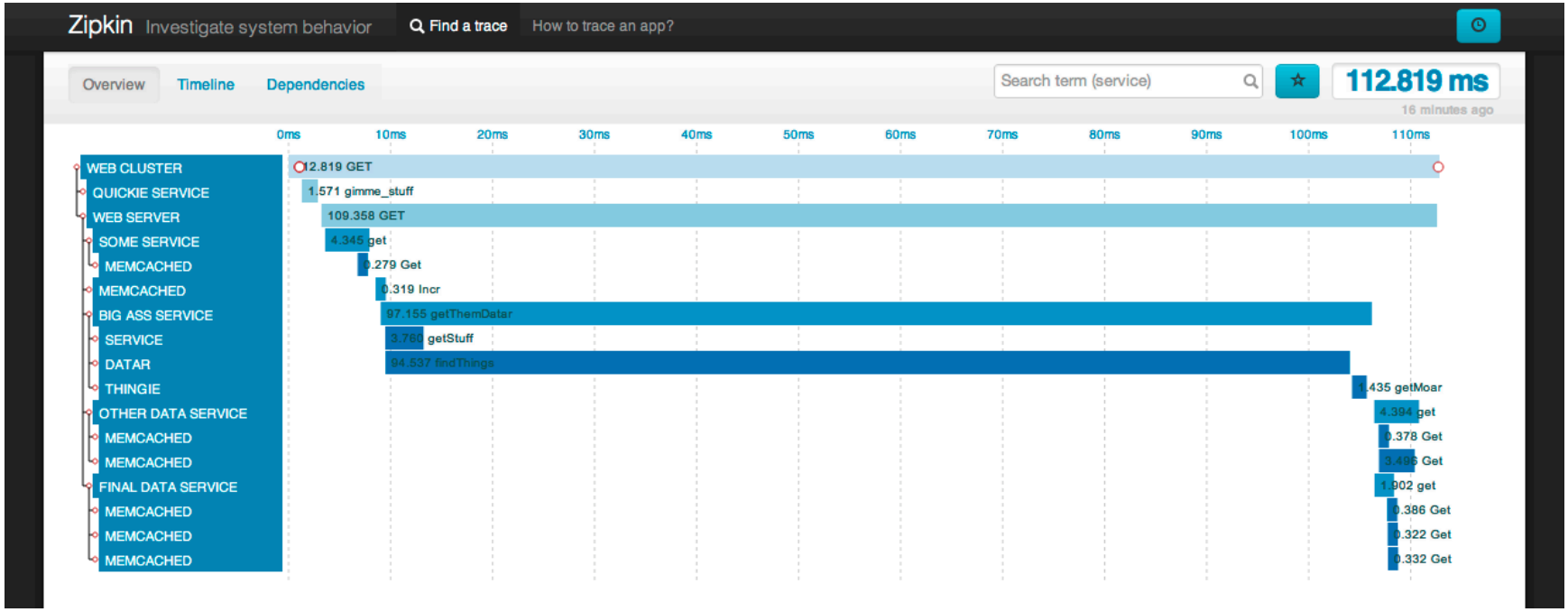
Annotation-Based	Black-box
More instrumentation	Less instrumentation
Less data is required to correlate	Depends on lot of data and statistical regression techniques
As granular as the user needs	Depends

Dapper: Requirements

- Not an experimental, fringe system
- Runs on production hardware and services
- Tracing

Annotation-Based	Black-box
More instrumentation	Less instrumentation
Less data is required to correlate	Depends on lot of data and statistical regression techniques
As granular as the user needs	Depends

Trace Trees and Spans



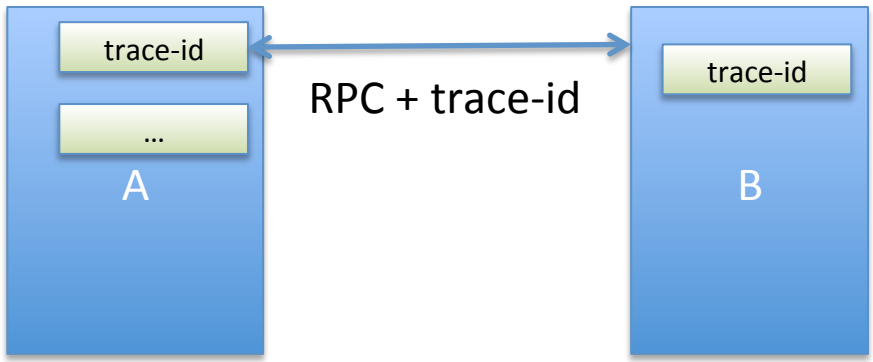
Trace Trees and Spans

- Spans are tree nodes with
  - Id
  - Parent Id
  - Name
  - Start and End times

  Can cover multiple hosts
- Edges indicate causal relationship
- Additional tier of infrastructure adds a level

Instrumenting

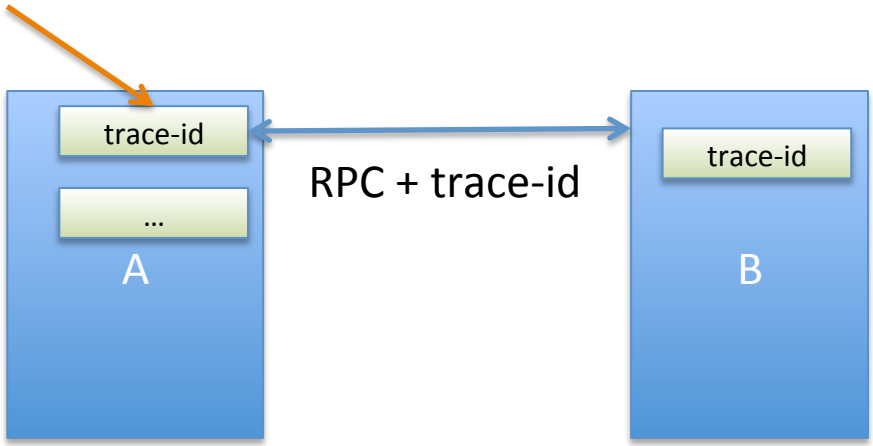
- Tracing the control path
  - Dapper attaches a thread-local *trace-context*
  - Async callbacks store the trace context of creator
    - Then associated by callee to the relevant worker thread



Instrumenting

- Tracing the control path
  - Dapper attaches a thread-local *trace-context*
  - Async callbacks store the trace context of creator
    - Then associated by callee to the relevant worker thread

Doesn't need to know about the actual thread in B.. Or aggregate up..



Annotations

- Application developers “enrich” code with annotations
- Allows map of custom key-value annotations

```
// 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.

Sampling

- Two levels
  - How many nodes in the cluster to sample
    - Runs as a daemon in fraction of production services
    - Only applications enriched with annotations are traced
  - Record only fraction of traces
    - On collector too
    - 1 TB of trace data / day => 0.01% of network traffic
    - Needs to be available for at least 2 weeks after initial logging

Implementation

- Very small and efficient implementation
  - No significant impact on network, disk or
- Traces are collected in log files in the machine
  - Then pulled and written to BigTable
  - Median Latency 15 seconds
    - But can take hours
  - There's a way to look at the logs live during "firefighting" operations too

Security & Privacy

- Payloads are not automatically logged
  - But you can opt-in if you need to

Tools

- Depot API
  - Expose clean intuitive interface to the raw data
  - Access By
    - Trace-Id
    - Bulk
    - Indexed Access
      - Compressed trace data is only 26% from actual trace data
- Web based UI for exposing trace trees

Tools

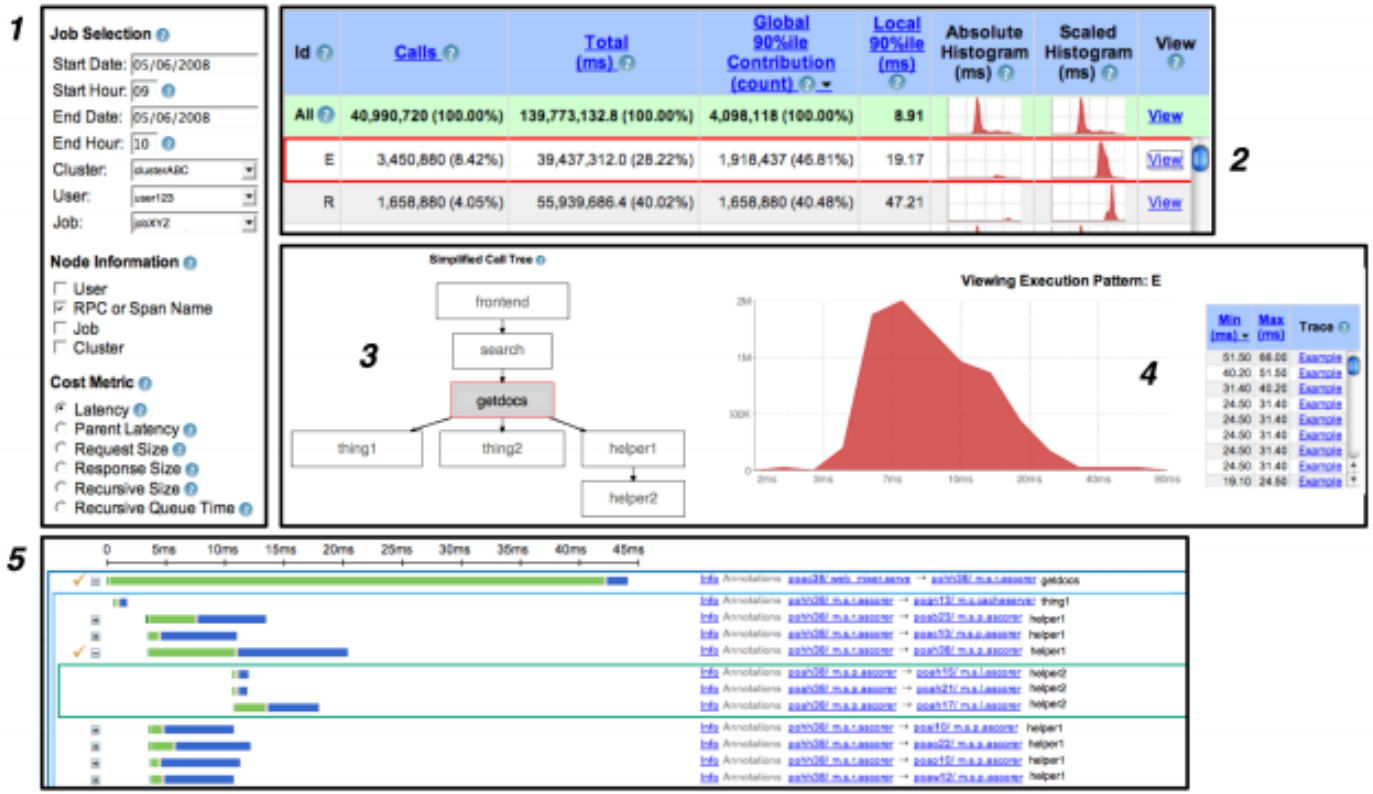


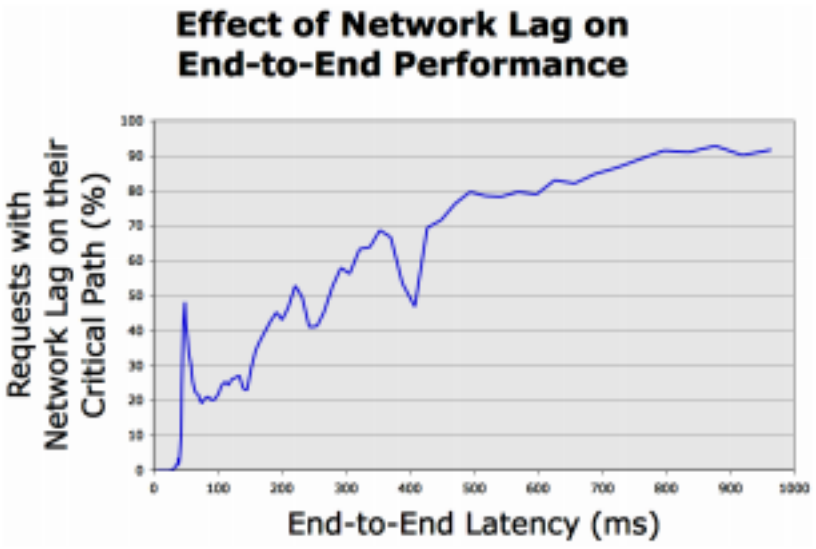
Figure 6: A typical user workflow in the general-purpose Dapper user interface.

Adoption

- 200 active users per week
- Teams use it to make informed refactoring decisions
  - Example AdWords
  - And add additional tools and extensions
- All data is exposed to build better tools and integrations
- Integrated with Exception Monitoring

Adoption

- Addressing Long-Tail Latency



Related Work

- Black-box monitoring systems
  - Project5
  - WAP5
  - Sherlock
- Explicit annotation based systems
  - Pip and Webmon rely on application level annotations
  - X-Trace, Pinpoint and Magpie focus on middleware and libraries modifications

Contributions

- First production distributed tracing framework
- Achieves higher degree of application level annotation
  - Very few workloads that required manual intervention (40 C++ and 33 Java applications)

Problems

- Specific to Google's infrastructure
- Batched workloads for efficiency do not have mapped trace-ids
  - MapReduce for example
- Good at system problems, you still need other tools for specifics
- Kernel levels are not included

Further

- Twitter implemented Zipkin inspired by Dapper
  - It's open source
- Enabled building distributed request flow comparison services at Google

Questions?