Dynamic Runtime Monitoring for Distributed Systems

Jonathan Mace
jcmace@cs.brown.edu

Ryan Roelke
rroelke@cs.brown.edu

Rodrigo Fonseca
rfonseca@cs.brown.edu

BROWN

Diagnosing distributed systems at runtime is **hard**!
Why? Problems in distributed systems are ***hard to predict*** and ***cross-component***

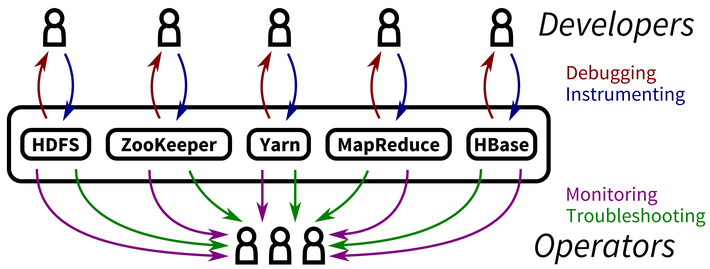
Problems are hard to predict:

- Cluster operators face unforeseen challenges and questions
- System monitoring tools are written by **system developers** at development time
- There is a *tug-of-war* over the tools, metrics and monitoring wanted by operators, and the tools developers are willing to provide

HDFS-5253

"*It'll be useful to have the requesting user's name ... for tracking per-user statistics*" - ***Operator***

"*I don't think we need to do this*" - ***Developer***

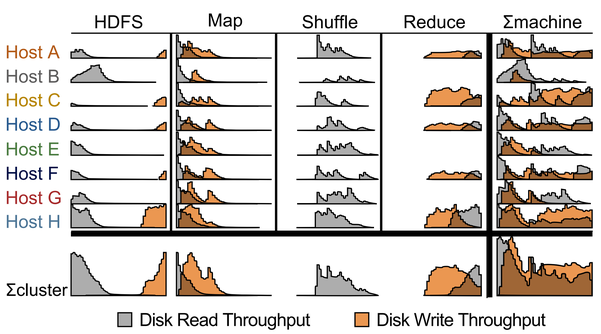


Problems are cross-component:

- Systems are designed to compose, so cross-component interactions are many and varied
- The root cause of an issue may occur in one component while symptoms manifest in another
- Monitoring and logging tools in today's systems **do not compose**

What do we want?

- Decide which events to monitor on the fly, at **runtime**
- Obtain arbitrary metrics at one point in the system while being able to select, filter, and group by events meaningful at other parts of the system

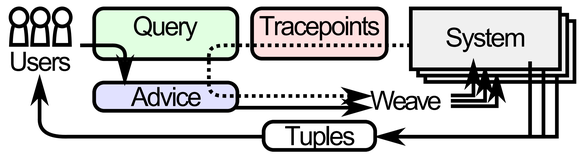


Disk operation metrics for a MapReduce sort job.

Metrics are broken down by host, source process, and operation type.

Systems today rarely perform cross-component correlations needed to provide metrics at this level of detail.

Pivot Tracing: a framework for dynamic, cross-component monitoring in Java

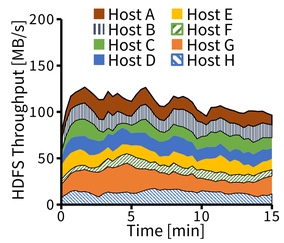


- Users can issue high-level queries to extract **real-time** statistics from the system
- Pivot Tracing enables **cross-component** analysis by forwarding partial query state along a request's execution path as it crosses thread and application boundaries
- Queries compile to **advice**, code that implements the query
- Pivot Tracing dynamically modifies the system to install advice ("**weave**")

Queries and Tracepoints

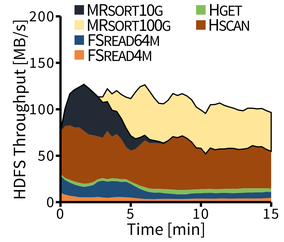
- A **tracepoint** is a location in the application's source code
  e.g., entry to any named function call
- A **query** can refer to one or more tracepoints and the variables that exist at those tracepoints
- The output to a query is a stream of **tuples**, produced when the tracepoints are invoked

```
From incr In DataNodeMetrics.incrBytesRead
GroupBy incr.host,
Select incr.host, SUM(incr.delta)
```



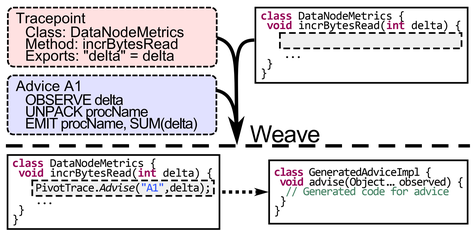
- A query can correlate multiple tracepoints with the *happened-before join* operation

```
From incr In DataNodeMetrics.incrBytesRead
 Join cl In First(ClientProtocols)
      On cl -> incr
 GroupBy cl.procName
 Select cl.procName SUM(incr.delta)
```



Dynamic Instrumentation

- Pivot Tracing weaves advice at tracepoints by modifying source code then hot-swapping the modified code
- Advice performs simple tuple operations, *e.g.* construct a tuple from tracepoint variables, emit an output tuple, *etc.*

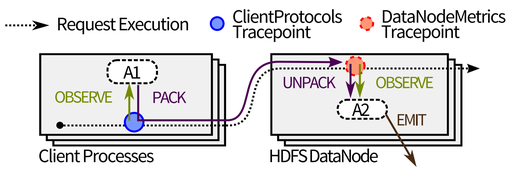


Zero-overhead instrumentation

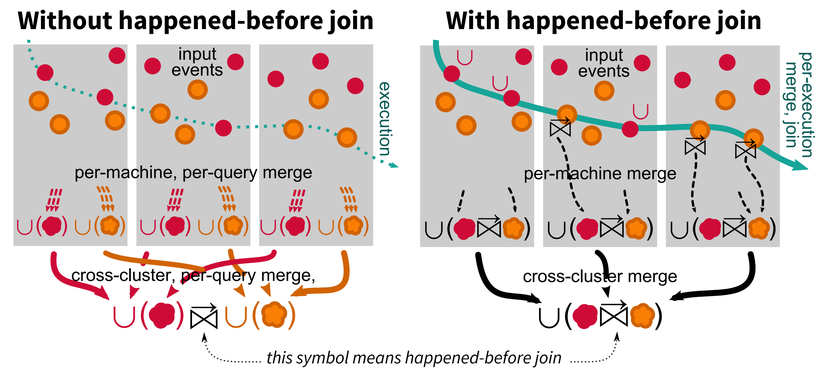
- Pivot Tracing does nothing by default except listen for new queries
- Pivot Tracing code runs only when a query is installed
- Installing a query has only a one-time cost to reload a class definition

Tracking Causality

- To implement happened-before join, Pivot Tracing propagates partial query state inline with a request, using causal metadata propagation techniques (like X-Trace)



- The expensive alternative to tracking causality is to gather data globally. Evaluating happened-before join during execution avoids the high cost of global aggregation

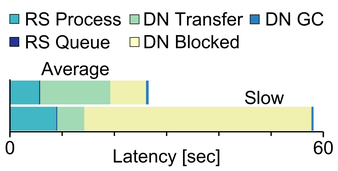


Pivot Tracing in action

- We have diagnosed several real-world problems using Pivot Tracing

Network Cable Limplock

A faulty network cable caused a network link downgrade from 1G to 100M. We used Pivot Tracing to decompose request latencies for anomalous requests. We found that the anomalous requests were blocking on network transfer at the faulty link.



HDFS Replica Selection bug

Clients to HDFS seemed to unduly favor certain replica hosts over others. We used Pivot Tracing to correlate clients with hosts. We found that clients always favored some hosts over others. Further investigation exposed the root cause to be faulty randomization logic when clients selected replicas.