# CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman

Fall 2018

# ADMINISTRIVIA

- Pick up papers after class or office hours
- Course Projects: two week targets

# MONITORING, DEBUGGING

# EXAMPLE SCENARIO

Setup: Cluster with HDFS, HBase, MapReduce

Goal: Monitor disk bandwidth used by each applications

Existing systems

- What gets recorded defined a priori
- No correlation across components
- e.g., only disk read throughput from DataNode

# CHALLENGES

Flexibility

- One size does not fit all ?
- Mismatch between developers and users
- Overhead of unused metrics

Cross-layer

- e.g., MapReduce on HBase on HDFS
- Need for end-to-end tracing

# APPROACH

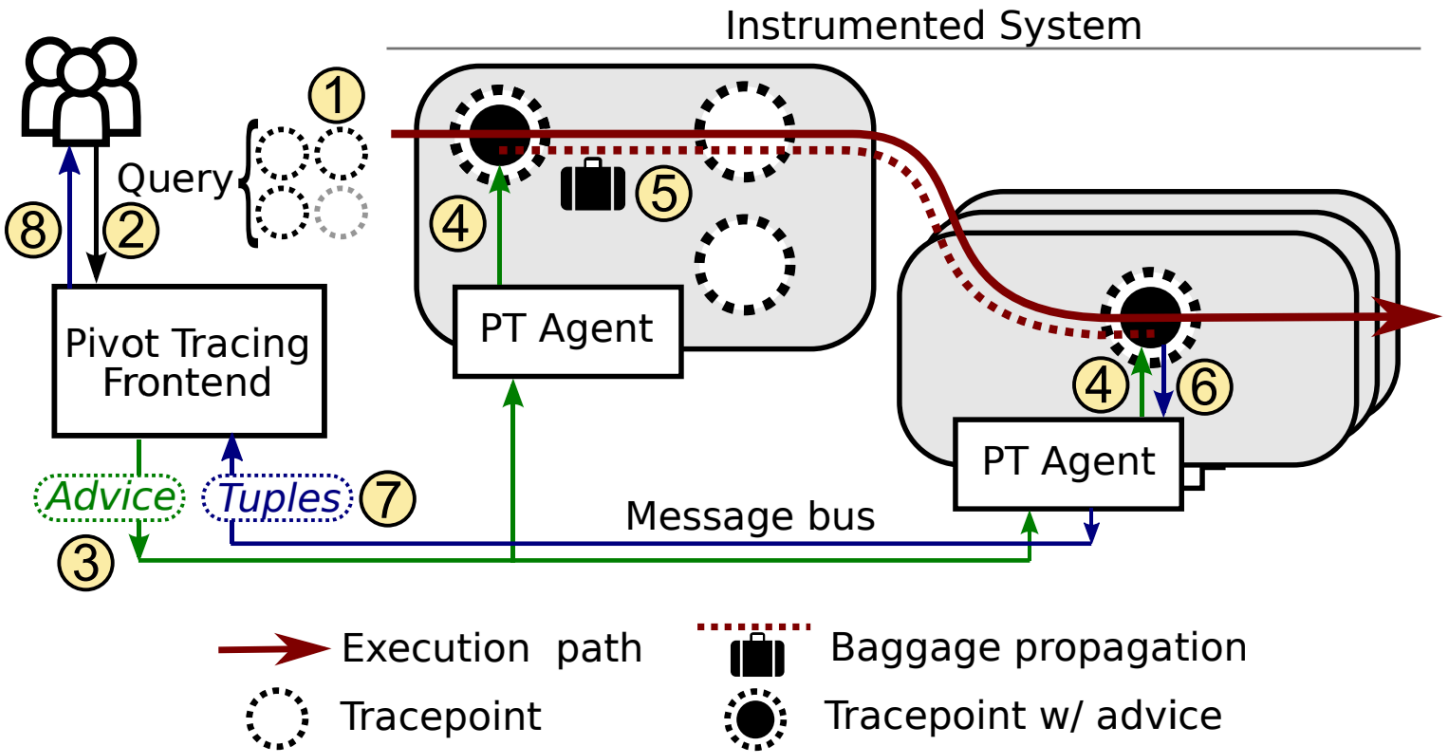
Tracepoints

- System developers define tracepoints
- Arbitrary interface / method signatures

Queries

- Events → Streaming, distributed data
- Compiled to IR advice
- Invoke advice every time tracepoint is triggered

# ARCHITECTURE



# DESIGN: TRACEPOINTS, QUERIES

Tracepoints

- Location in system code to instrument
- Export named vars, host, timestamp etc.
- Generate tuple

```
Tracepoint
   Class: DataNodeMetrics
   Method: incrBytesRead
   Exports: "delta" = delta
```

Query Language

- LINQ-style queries
- Selection, Projection
- Grouping, Aggregation etc.

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

# DESIGN: HAPPENED BEFORE

Happened before join

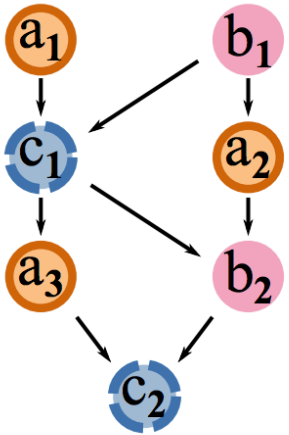
a → b if

a causally precedes b

for same request

$t_1 \in Q1, t_2 \in Q2$
such that $t_1 \rightarrow t_2$

Useful for root cause




*Query*

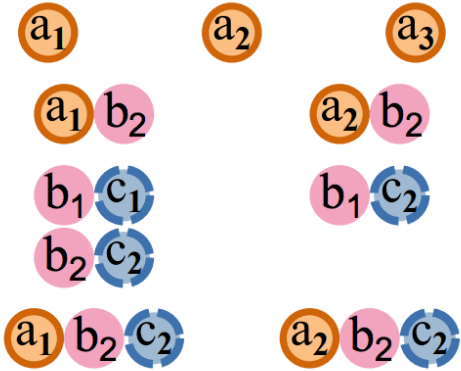
A

$A \overrightarrow{\bowtie} B$

$B \overrightarrow{\bowtie} C$

$(A \overrightarrow{\bowtie} B) \overrightarrow{\bowtie} C$

*Query Results*



# DESIGN: ADVICE

Advice: Intermediated representation for queries

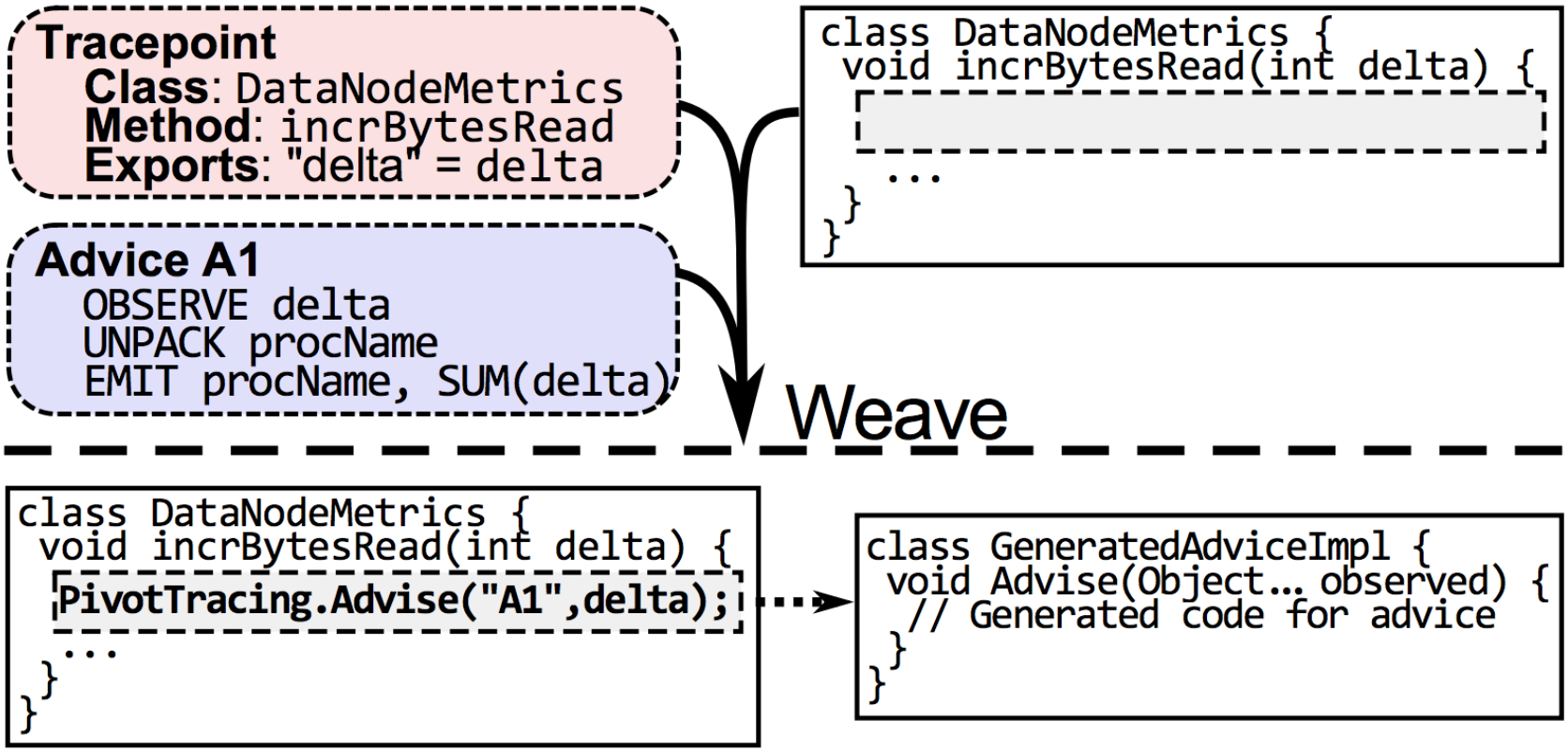
Executed at each tracepoint

No jumps or recursion

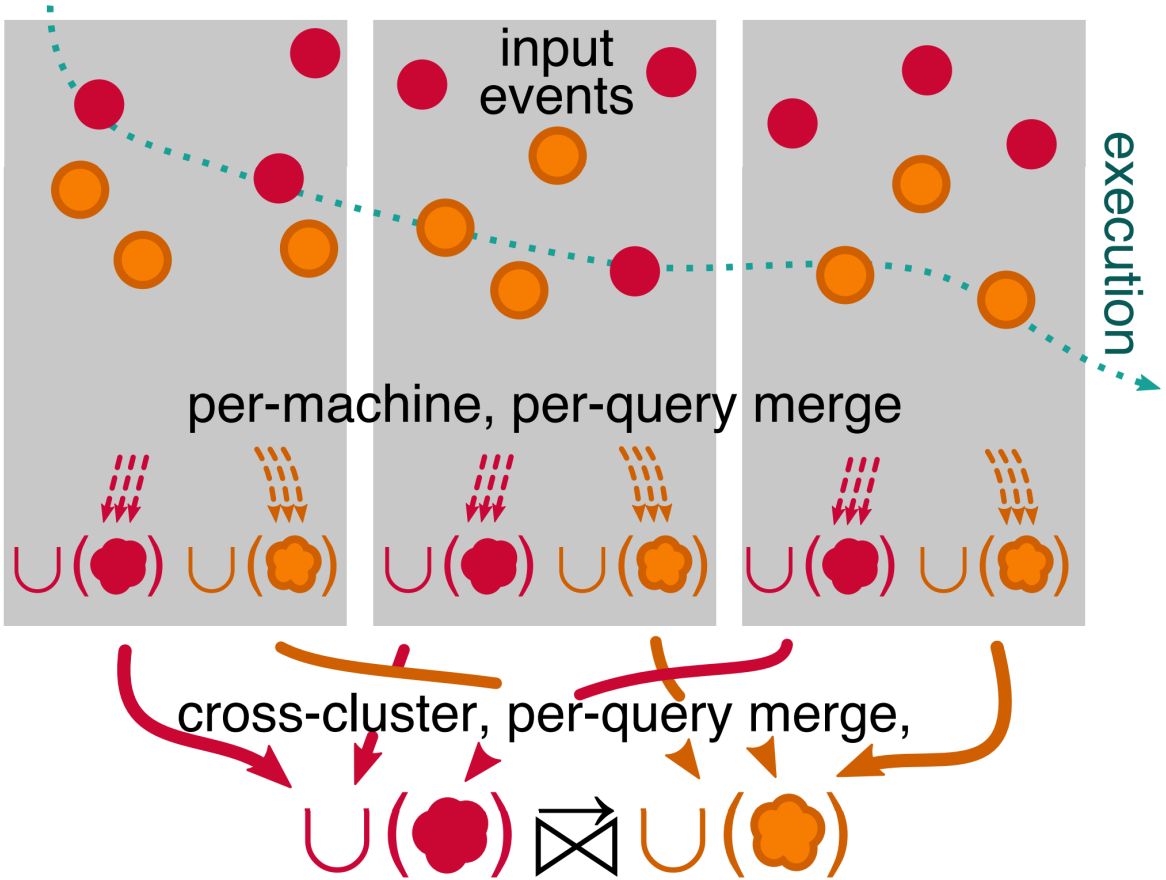
```
A2:OBSERVE delta
   UNPACK procName
   EMIT procName, SUM(delta)
```

| *Operation* | *Description* |
|---|---|
| OBSERVE | Construct a tuple from variables exported by a tracepoint |
| UNPACK | Retrieve one or more tuples from prior advice |
| FILTER | Evaluate a predicate on all tuples |
| PACK | Make tuples available for use by later advice |
| EMIT | Output a tuple for global aggregation |

# ADVICE EXECUTION



# OPTIMIZING JOINS: CHALLENGE



# HOW TO OPTIMIZE ?

Goal

- Reduce number of global tuples, num tuples packed

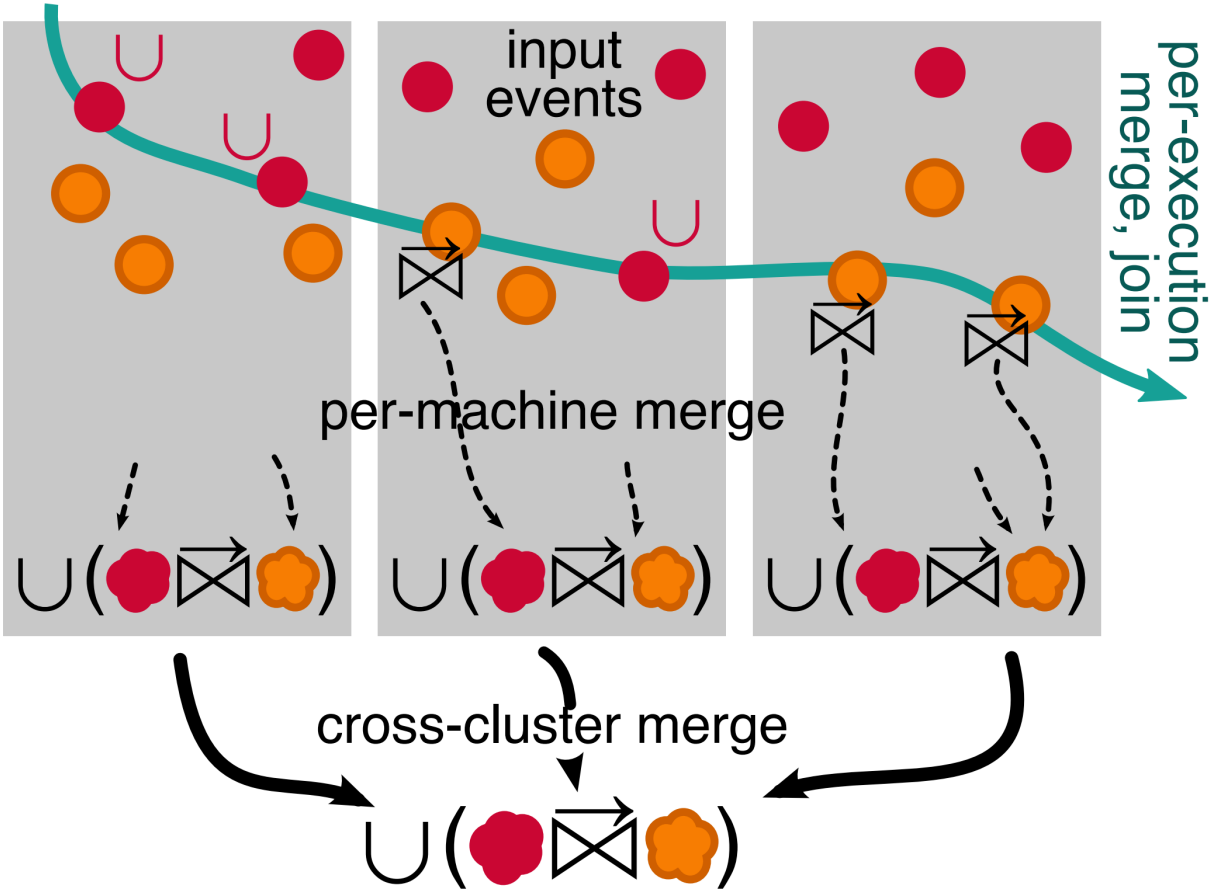
Baggage

- Per-request container for tuples
- Propagated alongside a request
- Automatically capture happens-before for joins

Other

- Push down aggregation, projection, filters etc.

# OPTIMIZED EXECUTION



# IMPLEMENTATION

Agents

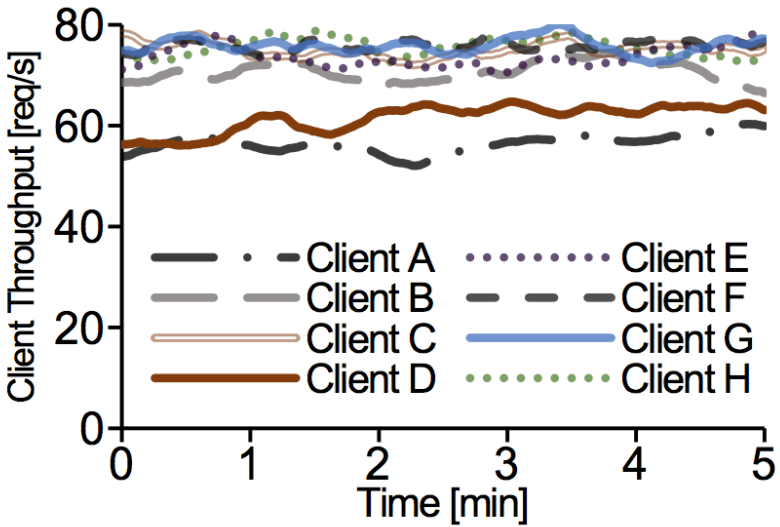
- Run thread in every process
- Export results every 1sec

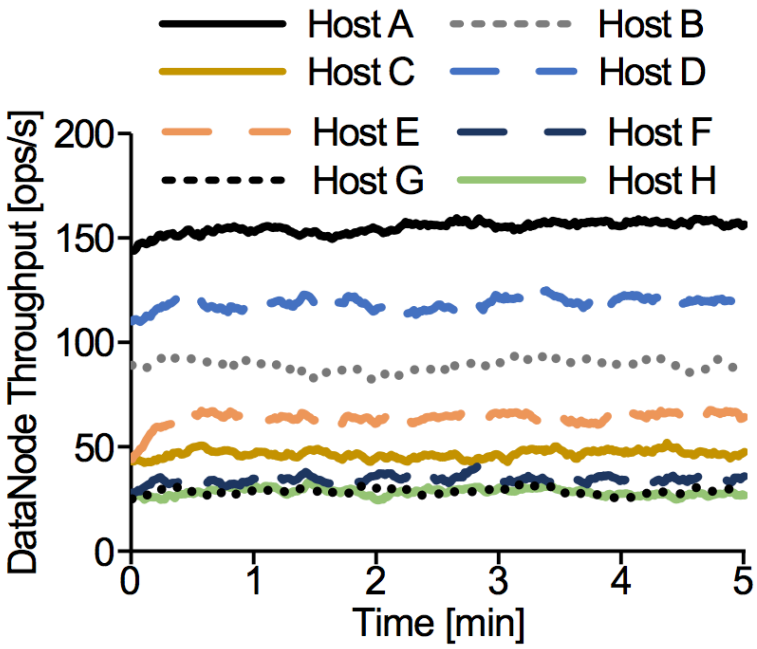
Advice, Tracing

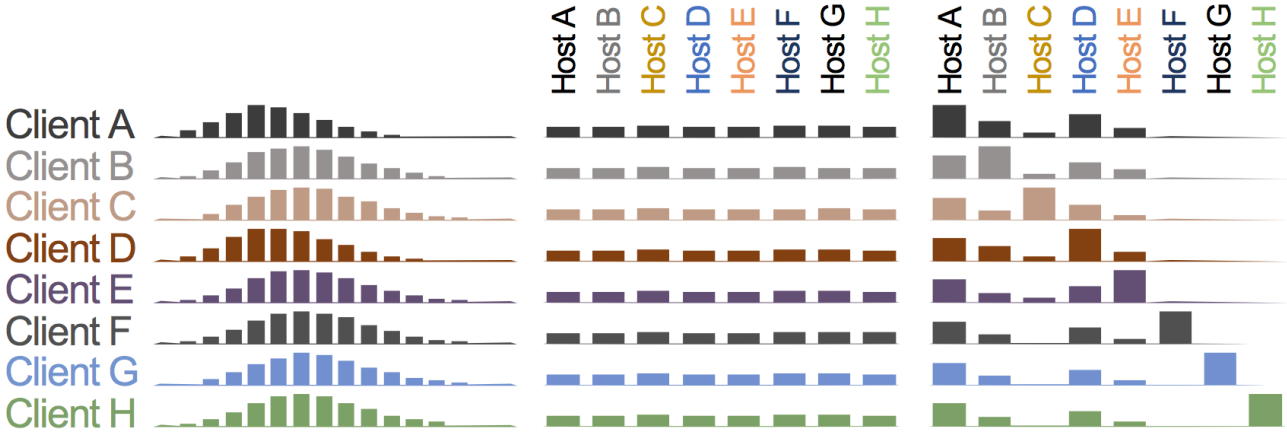
- Dynamically define tracepoints, advice
- Use `java.lang.instrument` – dynamic reload bytecode
- Zero overhead when no queries on tracepoint

Baggages sent as a part of RPC!

# CASE STUDY: HDFS REPLICA SELECTION






(d) Observed HDFS file read distribution (row) per client (col).

(e) Frequency each client (row) sees each DataNode (col) as a replica location.

(f) Frequency each client (row) subsequently selects each DataNode (col).

(g) Observed frequency of choosing one replica host (row) over another (col)

# SUMMARY

Importance of tracing for monitoring, debugging

Benefits

- Zero-overhead when unused
- Cross-application joins
- High-level queries

Drawbacks

- Need to instrument HDFS, Hadoop, HBase etc. ?
- How to pick queries that are installed ?