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 application

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
DESIGN: TRACEPOINTS, QUERIES

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

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

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

From incr In DataNodeMetrics.incrBytesRead
GroupBy incr.host
Select incr.host, SUM(incr.delta)
Happened before join

\[ a \rightarrow b \text{ if } a \text{ causally precedes } b \]

for same request

\[ t_1 \in Q_1, t_2 \in Q_2 \text{ such that } t_1 \rightarrow t_2 \]

Useful for root cause

<table>
<thead>
<tr>
<th>Execution Graph</th>
<th>Query</th>
<th>Query Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 ) \rightarrow ( c_1 ) \rightarrow ( a_2 ) \rightarrow ( b_1 )</td>
<td>( A \rightarrow B )</td>
<td>( a_1, a_2, a_3 )</td>
</tr>
<tr>
<td>( a_2 ) \rightarrow ( b_2 ) \rightarrow ( c_2 )</td>
<td>( B \rightarrow C )</td>
<td>( b_1, b_2, c_1, c_2 )</td>
</tr>
<tr>
<td>( (A \rightarrow B) \rightarrow C )</td>
<td></td>
<td>( a_1, b_2, c_2, a_2, b_2, c_2 )</td>
</tr>
</tbody>
</table>
Advice: Intermediated representation for queries
Executed at each tracepoint
No jumps or recursion

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBSERVE</td>
<td>Construct a tuple from variables exported by a tracepoint</td>
</tr>
<tr>
<td>UNPACK</td>
<td>Retrieve one or more tuples from prior advice</td>
</tr>
<tr>
<td>FILTER</td>
<td>Evaluate a predicate on all tuples</td>
</tr>
<tr>
<td>PACK</td>
<td>Make tuples available for use by later advice</td>
</tr>
<tr>
<td>EMIT</td>
<td>Output a tuple for global aggregation</td>
</tr>
</tbody>
</table>
ADVICE EXECUTION

**Tracepoint**
- **Class:** DataNodeMetrics
- **Method:** incrBytesRead
- **Exports:** "delta" = delta

**Advice A1**
- OBSERVE delta
- UNPACK procName
- EMIT procName, SUM(delta)

**Weave**

```java
class DataNodeMetrics {
    void incrBytesRead(int delta) {
        // Generated code for advice
        // ...
    }
}
```
OPTIMIZING JOINS: CHALLENGE

per-machine, per-query merge

input events

cross-cluster, per-query merge,

execution
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

input events

per-machine merge

per-execution merge, join

cross-cluster merge

U(● ◯)

U(● ◯)

U(● ◯)
IMPLEMENTATION

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

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

![Graph showing Client Throughput and DataNode Throughput over time](image)
(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?