Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Jonathan Mace, Ryan Roelke, Rodrigo Fonseca
Brown University
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Dynamically instrument live distributed systems
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Dynamically instrument live distributed systems

Correlate and group events across components
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Dynamically instrument live distributed systems

Correlate and group events across components

Hadoop Stack
Hadoop Stack
Hadoop Stack
Hadoop Stack
Hadoop Stack
Hadoop Stack
Hadoop Stack
How is disk bandwidth being used?
How is disk bandwidth being used?
How is disk bandwidth being used?
DataNode Metrics

HBase

MapReduce

HDFS

Disk

Machines

0 5 10 15
HDFS Throughput [MB/s]

Time [min]

0 50 100 150
MRSORT10G HSCAN
MRSORT100G HGET
FSREAD4M
FSREAD64M

Time [min]

0 5 10 15
HDFS Throughput [MB/s]
How is disk bandwidth being used?
How is disk bandwidth being used?
How is disk bandwidth being used?
How is disk bandwidth being used?
Instrumentation is decided at development time
Instrumentation is decided at development time

Probably not have enough info for your problem
Probably too much irrelevant info for your problem
Instrumentation is decided at development time

Probably not have enough info for your problem
Probably too much irrelevant info for your problem

Should every user bear the cost of a feature?
Instrumentation is decided at development time

Probably not have enough info for your problem
Probably too much irrelevant info for your problem

Should every user bear the cost of a feature?

HDFS-6292 Display HDFS per user and per group usage on webUI.
https://issues.apache.org/jira/browse/HDFS-6292
Instrumentation is decided at development time

Probably not have enough info for your problem
Probably too much irrelevant info for your problem

Should every user bear the cost of a feature?

HDFS-6292 Display HDFS per user and per group usage on webUI.
https://issues.apache.org/jira/browse/HDFS-6292
Dynamic dependencies

You often need to correlate information from different points in the system
Dynamic dependencies
You often need to correlate information from different points in the system

Systems are designed to compose

Systems don’t embed monitoring that relates to other services
Dynamic dependencies

You often need to correlate information from different points in the system

Systems are designed to compose

Systems don’t embed monitoring that relates to other services

Netflix “Death Star” Microservices Dependencies
@bruce_m_wong
Pivot Tracing
Pivot Tracing

You don’t know the questions in advance

Dynamic instrumentation

Fay (SOSP’11), Dtrace (ATC’04), ...
Pivot Tracing

You don’t know the questions in advance

Dynamic instrumentation
  Fay (SOSP’11), Dtrace (ATC’04), ...

You often need to correlate information from different points in the system

Causal tracing
  X-Trace (NSDI’07), Dapper (Google), Pip (NSDI’06), ...
Pivot Tracing
Pivot Tracing

Model system events as tuples in a streaming, distributed dataset
Pivot Tracing

Model system events as tuples in a streaming, distributed dataset

Dynamically evaluate relational queries over this dataset
Pivot Tracing

Model system events as tuples in a streaming, distributed dataset

Dynamically evaluate relational queries over this dataset

Happened-before Join (→)
Join based on Lamport’s happened-before relation
Pivot Tracing

Overview
public class DataNodeMetrics {
    ...  
    public void incrBytesRead(int delta) {
        ...  
    }  
    ...  
}
public class DataNodeMetrics {
    ...
    public void incrBytesRead(int delta) {
        ...
    } ...
}

("DataNodeMetrics", delta=10, host="hop01", ...)
DataNodeMetrics.java

```java
public class DataNodeMetrics {
    
    public void incrBytesRead(int delta) {
        
    }
}
```

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

(DataNodeMetrics”, delta=10, host=“hop01”, …)
From `incr` In `DataNodeMetrics.incrBytesRead`
GroupBy `incr.host`
Select `incr.host`, `SUM(incr.delta)`
From `incr` In `DataNodeMetrics.incrBytesRead`
GroupBy `incr.host`
Select `incr.host, SUM(incr.delta)`
```java
public class DataNodeMetrics {
    ...
    public void incrBytesRead(int delta) {
        ...
    }
    ...
}
```

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

```
("DataNodeMetrics", delta=10, host="hop01", ...)
```
public class DataNodeMetrics {
    ...  
    public void incrBytesRead(int delta) {
    ...
    }
    ...
}

Tracepoint
Class: DataNodeMetrics
Method: incrBytesRead
Exports: "delta"=delta
(“DataNodeMetrics”, delta=10, host="hop01", ...)
The diagram illustrates the integration of HBase, MapReduce, and HDFS with DataNode Metrics. The graph shows the throughput of HDFS in MB/s over time, with different data operations represented by distinct colors. The legend indicates operations such as MRSORT10G, HGET, FSREAD4M, and FSREAD64M.
Happened-before Join (:async)

Client Protocols

DataNode Metrics

HBase
MapReduce
HDFS

MRSORT10G HSCAN
MRSORT10G HGET
FSREAD4M
FSREAD64M

Happened-before Join (async)
Happened-before Join (.URI)
Happened-before Join (اعدة)

(Client Protocols, procName="HGET", ...)

(DataNode Metrics)

(Time [min] 0 5 10 15
HDFS Throughput [MB/s] Host A Host E
Host B Host F
Host C Host G
Host D Host H)

(Time [min] 0 5 10 15
HDFS Throughput [MB/s] Host A Host E
Host B Host F
Host C Host G
Host D Host H)

(MRSORT10G HSCAN
MRSORT10G HGET
FSREAD4M
FSREAD64M)

(ClientProtocols)

(Time [min] 0 5 10 15
HDFS Throughput [MB/s] Host A Host E
Host B Host F
Host C Host G
Host D Host H)

(MRSORT10G HSCAN
MRSORT10G HGET
FSREAD4M
FSREAD64M)
Happened-before Join (左手方向)

Client Protocols
(“ClientProtocols”, procName=“HGET”, …)

DataNode Metrics
(“DataNodeMetrics”, delta=10, host=“Hop01”, …)

HDFS Throughput [MB/s]
MRSORT10G HGET
MRSORT100G HSCAN
FSREAD4M
FSREAD64M

Time [min]
0 5 10 15

HBase
MapReduce
HDFS

Client Protocols
DataNode Metrics
Happened-before Join (mouseover)

(Client Protocols, procName=“HGET”, …)

(DataNodeMetrics, delta=10, host=“Hop01”, …)
Happened-before Join (<theaded>→</theaded>)

```plaintext
("ClientProtocols", procName="HGET", ...)

("DataNodeMetrics", delta=10, host="Hop01", ...)
```

---

![Diagram](image_url)
Happened-before Join (=DB)

(“ClientProtocols”, procName=“HGET”, ...)

(“DataNodeMetrics”, delta=10, host=“Hop01”, ...)

From incr In DataNodeMetrics.incrBytesRead
Join client In First(ClientProtocols) On client -> incr
GroupBy client.procName
Select client.procName, SUM(incr.delta)
Happened-before Join (∴)

```
(“DataNodeMetrics”, delta=10, host=“Hop01”, …)
(“ClientProtocols”, procName=“HGET”, …)
```

```
From incr In DataNodeMetrics.incrBytesRead
Join client In First(ClientProtocols) On client -> incr
GroupBy client.procName
Select client.procName, SUM(incr.delta)
```
Happened-before Join (▶)

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

Client Protocols

(“ClientProtocols”, procName=“HGET”, …)

DataNode Metrics

(“DataNodeMetrics”, delta=10, host=“Hop01”, …)
Happened-before Join (→)

From incr In DataNodeMetrics.incrBytesRead
Join client In First(ClientProtocols) On client -> incr
GroupBy client.procName
Select client.procName, SUM(incr.delta)
Happened-before Join (⟸)

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

(procname="HGET", delta=10, ...)
Design & Implementation
Pivot Tracing Pre-requisites
Design & Implementation
Pivot Tracing Pre-requisites

Dynamic instrumentation \(\rightarrow\) PT Agent
Design & Implementation
Pivot Tracing Pre-requisites

Dynamic instrumentation

PT Agent

Causal tracing

Baggage
Causal tracing

Baggage
Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

Causal tracing ➔ Baggage
Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request
Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request
Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request
Baggage is a Key:Value container propagated alongside a request

• Generalization of metadata in end-to-end tracing
• One instance per request
Baggage is a Key:Value container propagated alongside a request

- Generalization of metadata in end-to-end tracing
- One instance per request

Causal tracing

Baggage
Pivot Tracing Enabled
Design & Implementation
Queries
Instrumented System  (+ Baggage, + PT Agent)

PT Agent

PT Agent
Instrumented System (+ Baggage, + PT Agent)
Tracepoints

Places where PT can add instrumentation
Tracepoints

Places where PT can add instrumentation

Tracepoint A
Class: A
Method: A1()

Tracepoint B
Class: B
Method: B1()
Exports: “delta”=delta
**Tracepoints**

Places where PT can add instrumentation

Export identifiers accessible to queries
  - Defaults: host, timestamp, pid, proc name

**Tracepoint A**
- Class: A
- Method: A1()

**Tracepoint B**
- Class: B
- Method: B1()
- Exports: “delta”=delta
Tracepoints

Places where PT can add instrumentation

Export identifiers accessible to queries
  Defaults: host, timestamp, pid, proc name

Only references – not materialized until query is installed

Tracepoint A
  Class: A
  Method: A1()

Tracepoint B
  Class: B
  Method: B1()
  Exports: “delta”=delta
Query Language

Relational query language, similar to SQL, LINQ

- Selection
- Projection
- Filter
- GroupBy
- Aggregation
- Happened-Before Join

Refers to tracepoint-exported identifiers
Query Language

Relational query language, similar to SQL, LINQ

- Selection
- Projection
- Filter
- GroupBy
- Aggregation
- Happened-Before Join

Refers to tracepoint-exported identifiers

From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)

Tracepoint A
Class: A
Method: A1()

Tracepoint B
Class: B
Method: B1()
Exports: “delta”=delta
Query Language

Relational query language, similar to SQL, LINQ

- Selection
- Projection
- Filter
- GroupBy
- Aggregation
- Happened-Before Join

Refers to tracepoint-exported identifiers

Output: stream of tuples
  e.g., (procName, delta)
Advice

Query is compiled to advice
(intermediate representation for instrumentation)
Instrumented System (+ Baggage, + PT Agent)

Advice

Query is compiled to advice
(intermediate representation for instrumentation)

Advice will be installed at tracepoints

```
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
```

Advice A1
OBSERVE procName
PACK procName

Advice B1
OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)
Advice

Query is compiled to advice (intermediate representation for instrumentation)

Advice will be installed at tracepoints

Limited instruction set

- OBSERVE
- PACK
- FILTER
- UNPACK
- EMIT

```
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
```
PT Agent dynamically enables advice at tracepoints.

Advice A1:
- OBSERVE procName
- PACK procName

Advice B1:
- OBSERVE delta
- UNPACK procName
- EMIT procName, SUM(delta)

Query:
```
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
```
Instrumented System (+Baggage, +PT Agent)

Evaluating

From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)

Advice A1
OBSERVE procName
PACK procName

Advice B1
OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)

Advice A1
OBSERVE procName
PACK procName

Advice B1
OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)

Evaluating
From A
Join B On a.procName -> b.procName
GroupBy a.procName
Select a.procName, SUM(b.delta)

Advice A1
| OBSERVE procName
| PACK procName

Advice B1
| OBSERVE delta
| UNPACK procName
| EMIT procName, SUM(delta)
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)

Advice A1
OBSERVE procName
PACK procName

Advice B1
OBSERVE delta
UNPACK procName
EMIT procName, SUM(delta)
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
From A In A
Join B In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
From $a$ In $A$
Join $b$ In $B$ On $a$ -> $b$
GroupBy $a$.procName
Select $a$.procName, $\text{SUM}(b\.delta)$

Advice A1
OBSERVE procName
PACK procName

Advice B1
OBSERVE delta
UNPACK procName
EMIT procName, $\text{SUM}(\text{delta})$
Evaluating

Baggage explicitly follows execution
Evaluated inline during a request
(no global aggregation needed)
Query Results

Tuples are accumulated locally in PT Agent

```
From a In A
Join b In B On a -> b
GroupBy a.procName
Select a.procName, SUM(b.delta)
```
Query Results

Tuples are accumulated locally in PT Agent

Periodically reported back to user
  e.g., every second
Pivot Tracing
Evaluation
Java-Based Implementation
Java-Based Implementation

PT agent thread that runs inside each process
- Javassist for dynamic instrumentation
- PubSub to receive commands / send tuples
Java-Based Implementation

PT agent thread that runs inside each process
- Javassist for dynamic instrumentation
- PubSub to receive commands / send tuples

Baggage library for use by instrumented system
- Data format specified using Protocol Buffers
Java-Based Implementation

- **PT Agent**: PT agent thread that runs inside each process
  - Javassist for dynamic instrumentation
  - PubSub to receive commands / send tuples

- **Baggage library**: for use by instrumented system
  - Data format specified using Protocol Buffers

- **Front-end client library**
  - Define tracepoints and write text queries
  - Compile queries to advice
  - Submit advice to PT agents
Pivot Tracing Enabled  (+ Baggage, + PT Agent)
Adding Baggage: ~50-200 lines of code per system
Adding Baggage: ~50-200 lines of code per system

Primarily modifying execution boundaries:
  Thread, Runnable, Callable, Queue
  RPC invocations
Adding Baggage: ~50-200 lines of code per system

Primarily modifying execution boundaries:
- Thread, Runnable, Callable, Queue
- RPC invocations
Pivot Tracing Overheads

• Pivot Tracing Enabled  (+леч Baggage, + PT Agent)
  Application level benchmarks: baseline 0.3% overhead
Pivot Tracing Overheads

• Pivot Tracing Enabled  (+(@(Baggage, + @(PT Agent)
 Application level benchmarks: baseline 0.3% overhead

• No overhead for queries / tracepoints until installed
Pivot Tracing Overheads

• Pivot Tracing Enabled  (+ Baggage, + PT Agent)
  Application level benchmarks: baseline 0.3% overhead

• No overhead for queries / tracepoints until installed

• With queries from paper installed
  Application level benchmarks: max 14.3% overhead
  (CPU-only lookups)
Pivot Tracing Overheads

• Pivot Tracing Enabled  (+ Bags, + PT Agent)
  Application level benchmarks: **baseline** 0.3% overhead

• No overhead for queries / tracepoints until installed

• With queries from paper installed
  Application level benchmarks: max 14.3% overhead
  (CPU-only lookups)
  Largest baggage size: ~137 bytes
Experiments
Experiments

1. Monitoring queries with various groupings
Experiments

1. Monitoring queries with various groupings

2. Decomposing request latencies
Experiments

1. Monitoring queries with various groupings
2. Decomposing request latencies
3. Debugging recurring problems
Experiments

1. Monitoring queries with various groupings
2. Decomposing request latencies
3. Debugging recurring problems
HDFS
NameNode
Filesystem Metadata
HDFS
NameNode

Replicated block storage

Filesystem Metadata
HDFS
DataNode
Filesystem Metadata
HDFS NameNode
Replicated block storage
HDFS

NameNode

File

File

File

File

Replicated block storage
HDFS
NameNode

Filesystem Metadata

File
2 3 5

Replicated block storage
HDFS NameNode

File 2 3 5

GetBlockLocations

Filesystem Metadata

Client

Replicated block storage
HDFS Replica Selection Policy

- Same machine? → read local
- Same rack? → read rack-local
- Otherwise → select randomly

Filesystem Metadata

HDFS
NameNode

Client

GetBlockLocations

File 2 3 5

Replicated block storage

HDFS DataNode

HDFS DataNode

HDFS DataNode

HDFS DataNode

HDFS DataNode
HDFS Replica Selection Policy

- Same machine? -> read local
- Same rack? -> read rack-local
- Otherwise -> select randomly
Client Workload Generator

- Randomly read from large dataset

HDFS DataNode

8 Worker Hosts
8 Worker Hosts

- Client Workload Generator
  - Randomly read from large dataset
- HDFS DataNode

+ HDFS NameNode
Same machines, same processes, same workloads
Same machines, same processes, same workloads

I expected uniform throughput from workload generators
Same machines, same processes, same workloads

I expected uniform throughput from workload generators

Consistently lower throughput
Same machines, same processes, same workloads

I expected uniform throughput from workload generators
I expected uniform throughput on DataNodes
Same machines, same processes, same workloads

I expected uniform throughput from workload generators
I expected uniform throughput on DataNodes

[Graphs showing variations in throughput over time]
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files
It’s probably a bug in the workload generator I wrote

My hypothesis:
   Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
   GroupBy blockLocations.fileName
   Select blockLocations.fileName, COUNT
```
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```
From blockLocations In NameNode.GetBlockLocations
GroupBy blockLocations.fileName
Select blockLocations.fileName, COUNT
```

![Graph showing frequency of file access](image-url)
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

From blockLocations In NameNode.GetBlockLocations
Join cl In Client.DoRandomRead On cl -> blockLocations
GroupBy cl.host, blockLocations.fileName
Select cl.host, blockLocations.fileName, COUNT

Frequency

Number of times accessed
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

From `blockLocations` In `NameNode.GetBlockLocations`
Join `cl` In `Client.DoRandomRead` On `cl` -> `blockLocations`
GroupBy `cl.host`, `blockLocations.fileName`
Select `cl.host`, `blockLocations.fileName`, COUNT

Frequency

Number of times accessed
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

From blockLocations In NameNode.GetBlockLocations
Join cl In Client.DoRandomRead On cl -> blockLocations
GroupBy cl.host, blockLocations.fileName
Select cl.host, blockLocations.fileName, COUNT

Frequency

Number of times accessed
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

```sql
From blockLocations In NameNode.GetBlockLocations
Join cl In Client.DoRandomRead On cl -> blockLocations
GroupBy cl.host, blockLocations.fileName
Select cl.host, blockLocations.fileName, COUNT
```

![Diagram showing HDFS NameNode, GetBlockLocations, and Client DoRandomRead operations with a graph of frequency vs. number of times accessed.](image-url)
It’s probably a bug in the workload generator I wrote

My hypothesis:

Workload generator is not randomly looking up files

From blockLocations In NameNode.GetBlockLocations
Join cl In Client.DoRandomRead On cl -> blockLocations
GroupBy cl.host, blockLocations.fileName
Select cl.host, blockLocations.fileName, COUNT
It’s probably a bug in the workload generator I wrote

My hypothesis:
Workload generator is not randomly looking up files

From `blockLocations In NameNode.GetBlockLocations`
Join `cl In Client.DoRandomRead` On `cl -> blockLocations`
GroupBy `cl.host, blockLocations.fileName`
Select `cl.host, blockLocations.fileName, COUNT`
Maybe skewed DataNode throughput is because some DataNodes store more files than others
Maybe skewed DataNode throughput is because some DataNodes store more files than others.

How often was each DataNode a replica host?
Maybe skewed DataNode throughput is because some DataNodes store more files than others.

How often was each DataNode a replica host?

From blockLocations In NameNode.GetBlockLocations
GroupBy blockLocations.replicas
Select blockLocations.replicas, COUNT
Maybe skewed DataNode throughput is because some DataNodes store more files than others

How often was each DataNode a replica host?

From blockLocations In NameNode.GetBlockLocations
GroupBy blockLocations.replicas
Select blockLocations.replicas, COUNT

[Chart showing replica location and count for DataNodes A to H]
Maybe skewed DataNode throughput is because some DataNodes store more files than others

How often was each DataNode a replica host?

From blockLocations In NameNode.GetBlockLocations
Join cl In Client.DoRandomRead On cl -> blockLocations
GroupBy cl.host, blockLocations.replicas
Select cl.host, blockLocations.replicas, COUNT
Maybe skewed DataNode throughput is because some DataNodes store more files than others.

How often was each DataNode a replica host?

```
From blockLocations In NameNode.GetBlockLocations
Join cl In Client.DoRandomRead On cl -> blockLocations
GroupBy cl.host, blockLocations.replicas
Select cl.host, blockLocations.replicas, COUNT
```
Conclusions so far:

- Clients are selecting files uniformly at random
- Files are distributed across DNs uniformly at random
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random
Conclusions so far:
   Clients are selecting files uniformly at random
   Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else *could* it have been read from?

---

<table>
<thead>
<tr>
<th>HDFS Replica Selection Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine?</td>
</tr>
<tr>
<td>Same rack?</td>
</tr>
<tr>
<td>Otherwise</td>
</tr>
</tbody>
</table>
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else could it have been read from?

HDFS Replica Selection Policy
- Same machine? -> read local
- Same rack? -> read rack-local
- Otherwise -> select randomly
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else could it have been read from?

From readBlock In DataNode.DataTransferProtocol
Join blockLocations In NameNode.GetBlockLocations
On blockLocations -> readBlock
GroupBy blockLocations.replicas, readBlock.host
Select blockLocations.replicas, readBlock.host, COUNT
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else *could* it have been read from?

From `readBlock In DataNode.DataTransferProtocol`
Join `blockLocations In NameNode.GetBlockLocations`
On `blockLocations` -> `readBlock`
GroupBy `blockLocations.replicas, readBlock.host`
Select `blockLocations.replicas, readBlock.host, COUNT`
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else could it have been read from?

From `readBlock In DataNode.DataTransferProtocol`  
Join `blockLocations In NameNode.GetBlockLocations`  
On `blockLocations` -> `readBlock`  
GroupBy `blockLocations.replicas, readBlock.host`  
Select `blockLocations.replicas, readBlock.host, COUNT`
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else could it have been read from?

From readBlock In DataNode.DataTransferProtocol
Join blockLocations In NameNode.GetBlockLocations
    On blockLocations -> readBlock
Join cl In Client.DoRandomRead
    On cl -> blockLocations
Where cl.host != readBlock.host
GroupBy blockLocations.replicas, readBlock.host
Select blockLocations.replicas, readBlock.host, COUNT
Conclusions so far:
Clients are selecting files uniformly at random
Files are distributed across DNs uniformly at random

Hypothesis: choice of replica isn’t random?

When a file is read from a DataNode, where else *could* it have been read from?

From `readBlock` In `DataNode.DataTransferProtocol`
Join `blockLocations` In `NameNode.GetBlockLocations`
  On `blockLocations` -> `readBlock`
Join `cl` In `Client.DoRandomRead`
  On `cl` -> `blockLocations`
Where `cl.host` != `readBlock.host`
GroupBy `blockLocations.replicas`, `readBlock.host`
Select `blockLocations.replicas`, `readBlock.host`, COUNT

HDFS Replica Selection Policy
- Same machine? -> read local
- Same rack? -> read rack-local
- Otherwise -> select randomly
From readBlock In DataNode.DataTransferProtocol
Join blockLocations In NameNode.GetBlockLocations
   On blockLocations -> readBlock
Join cl In Client.DoRandomRead
   On cl -> blockLocations
Where cl.host != readBlock.host
GroupBy blockLocations.replicas, readBlock.host
Select blockLocations.replicas, readBlock.host, COUNT
When both B and D host replicas, Clients choose B this often: (~50%) Clients choose D this often: (~50%)

From readBlock In DataNode.DataTransferProtocol
Join blockLocations In NameNode.GetBlockLocations
    On blockLocations -> readBlock
Join cl In Client.DoRandomRead
    On cl -> blockLocations
Where cl.host != readBlock.host
GroupBy blockLocations.replicas, readBlock.host
Select blockLocations.replicas, readBlock.host, COUNT
When both $\mathcal{A}$ and $\mathcal{C}$ host replicas, Clients choose $\mathcal{A}$ this often: $\square$ (100%)

From $\text{readBlock}$ In $\text{NameNode.DataTransferProtocol}$

Join $\text{blockLocations}$ In $\text{NameNode.GetBlockLocations}$

On $\text{blockLocations}$ -> $\text{readBlock}$

Join $\text{cl}$ In $\text{Client.DoRandomRead}$

On $\text{cl}$ -> $\text{blockLocations}$

Where $\text{cl.host} \neq \text{readBlock.host}$

GroupBy $\text{blockLocations.replicas, readBlock.host}$

Select $\text{blockLocations.replicas, readBlock.host, COUNT}$

HDFS Replica Selection Policy

- Same machine? -> read local
- Same rack? -> read rack-local
- Otherwise -> select randomly
When both \( A \) and \( C \) host replicas, Clients choose \( A \) this often: \( ✔ \) (100%)  
Clients choose \( C \) this often: \( \_ \) (0%)
When both A and C host replicas, Clients choose A this often: 100% (100%)
Clients choose C this often: 0% (0%)

From `readBlock In DataNode.DataTransferProtocol`
Join `blockLocations In NameNode.GetBlockLocations` On `blockLocations -> readBlock`
Join `cl In Client.DoRandomRead` On `cl -> blockLocations`
Where `cl.host != readBlock.host`
GroupBy `blockLocations.replicas, readBlock.host`
Select `blockLocations.replicas, readBlock.host, COUNT`
When both B and D host replicas, Clients choose B this often: _ (0%)
When both B and D host replicas, Clients choose B this often: ___ (0%)
When both ♛ and ♞ host replicas, Clients choose ♛ this often: (0%)
Clients choose ♞ this often: (100%)

From readBlock In DataNode.DataTransferProtocol
Join blockLocations In NameNode.GetBlockLocations
  On blockLocations -> readBlock
Join cl In Client.DoRandomRead
  On cl -> blockLocations
Where cl.host != readBlock.host
GroupBy blockLocations.replicas, readBlock.host
Select blockLocations.replicas, readBlock.host, COUNT
When both B and D host replicas, Clients choose B this often: (0%) Clients choose D this often: (100%)
When both B and D host replicas, Clients choose B this often: (0%) Clients choose D this often: (100%)

From readBlock In DataNode.DataTransferProtocol
Join blockLocations In NameNode.GetBlockLocations
  On blockLocations -> readBlock
Join cl In Client.DoRandomRead
  On cl -> blockLocations
Where cl.host != readBlock.host
GroupBy blockLocations.replicas, readBlock.host
Select blockLocations.replicas, readBlock.host, COUNT
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data Node Throughput (ops/s)

Time [min]

HDFS Replica Selection Policy

- Same machine? → read local
- Same rack? → read rack-local
- Otherwise → select randomly
• Lack of randomization skewed workload toward certain DNs
- Lack of randomization skewed workload toward certain DNs
- Independently discovered. Fixed in HDFS 2.5

**HDFS Replica Selection Policy**

- Same machine?
  - read local
- Same rack?
  - read rack-local
- Otherwise
  - select randomly
• Lack of randomization skewed workload toward certain DNs
• Independently discovered. Fixed in HDFS 2.5

• Seamlessly add correlations between multiple components
• Very specific, one-off metrics
• This experiment: 1.5% application-level overhead
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Happened-Before Join
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Happened-Before Join

Dynamic Instrumentation  Causal Tracing
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Happened-Before Join

Dynamic Instrumentation  Causal Tracing

Acceptable overheads for production (we think)
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Happened-Before Join

Dynamic Instrumentation  Causal Tracing

Acceptable overheads for production (we think)
Standing basic queries    Potential to dig deeper
Pivot Tracing
Dynamic Causal Monitoring for Distributed Systems

Acceptable overheads for production (we think)
Standing basic queries Potential to dig deeper

Jonathan Mace Ryan Roelke Rodrigo Fonseca