COMPUWARE APM

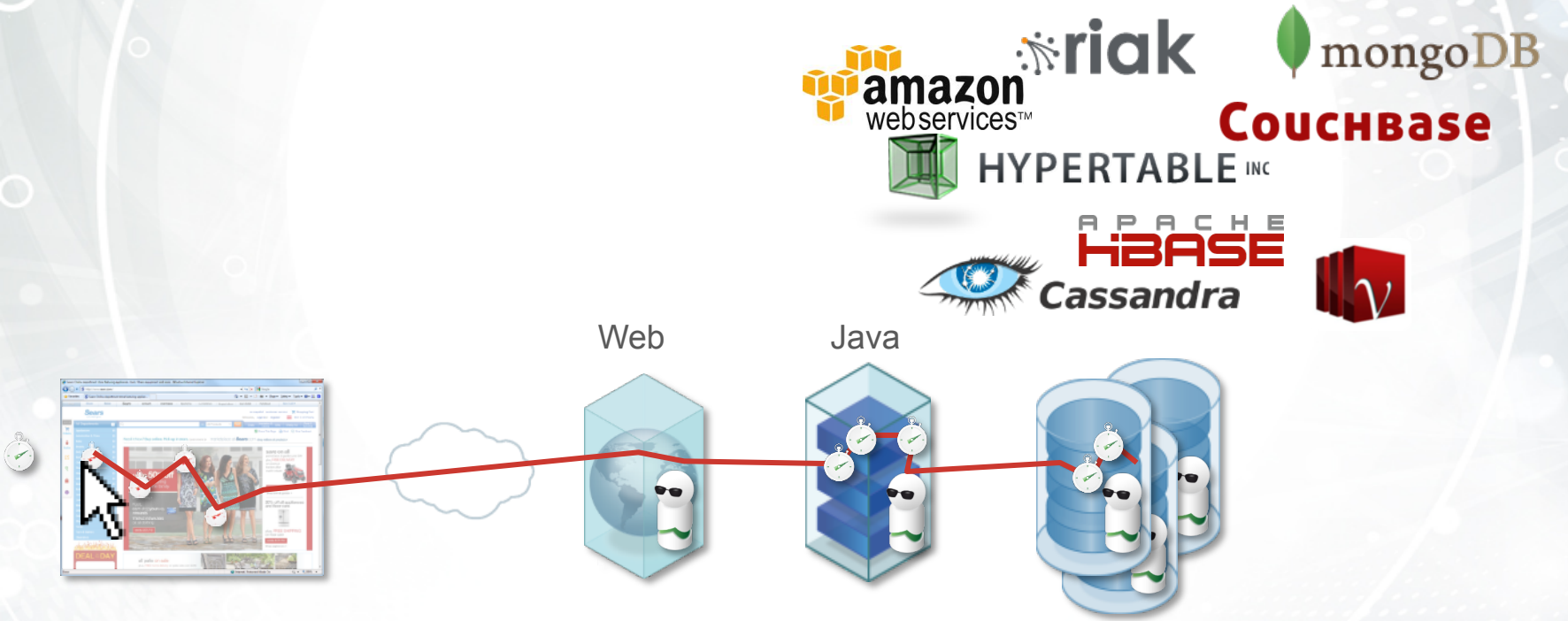
DYNATRACE® | GOMEZ®

# Performance Management in 'Big Data' Applications

Michael Kopp, Technology Strategist

@mikopp

# NoSQL: High Volume/Low Latency DBs



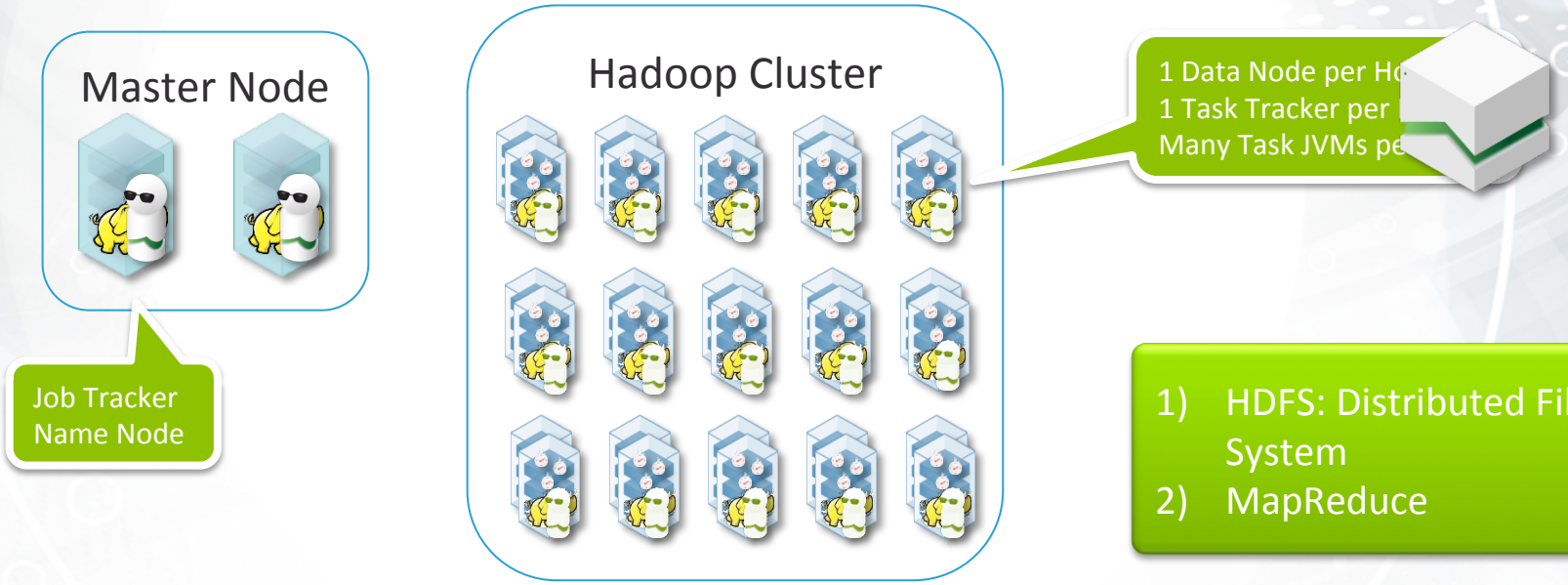

**Key Challenges**

1) Even Distribution
2) Correct Schema and Access patterns
3) Understanding Application Impact
4) Proprietary API

**Key Benefits**

1) Fast Read/Write
2) Horizontal Scalability
3) Redundancy and High Availability

# Hadoop: Large Scale Parallel Processing




1) HDFS: Distributed File System
2) MapReduce

**Key Challenges**

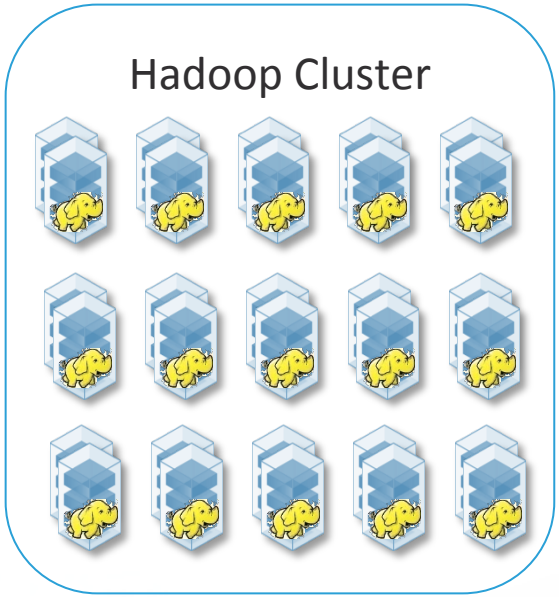
1) Optimal Distribution
2) Unwieldy Configuration
3) Can easily waste your resources
4) Failure or Error Analysis is hard
5) Performance Optimization is hard

**Key Benefits**

1) Massive Horizontal Batch Job
2) Split big Problems into smaller ones

# Raw Data ingest

Ingest Data



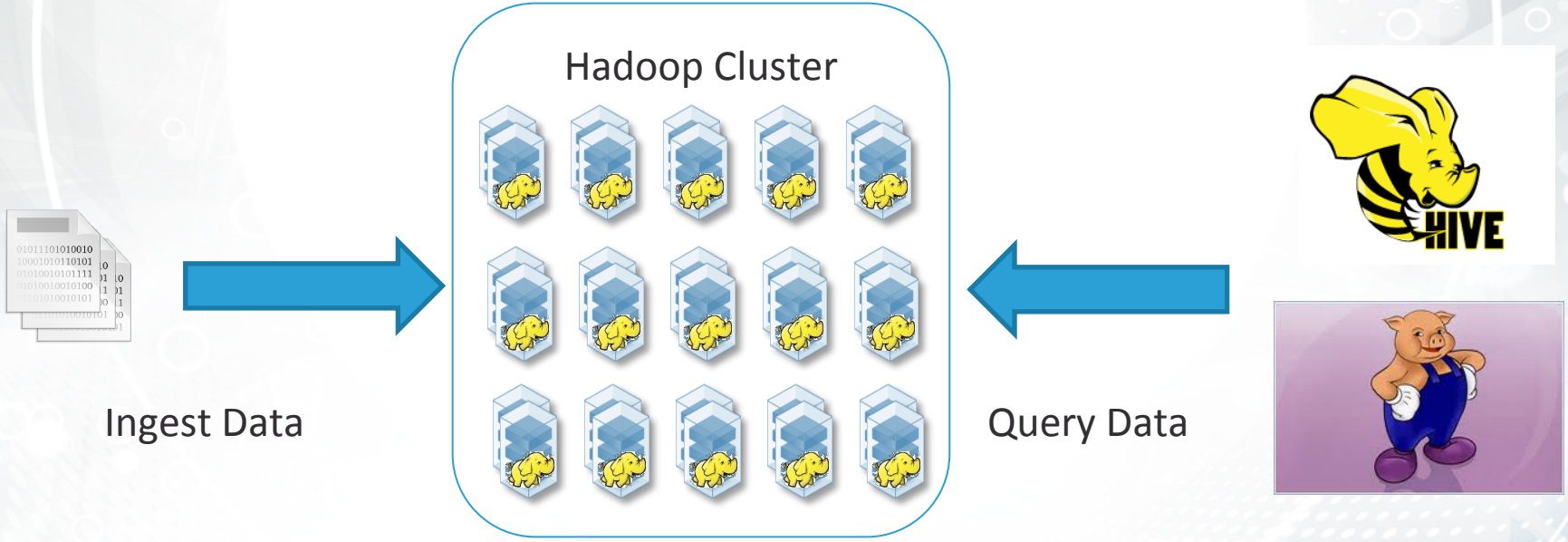

Query Data



**Raw Data!**

**Slow and Cumbersome**

# Transform to Hive/Pig Structured Data



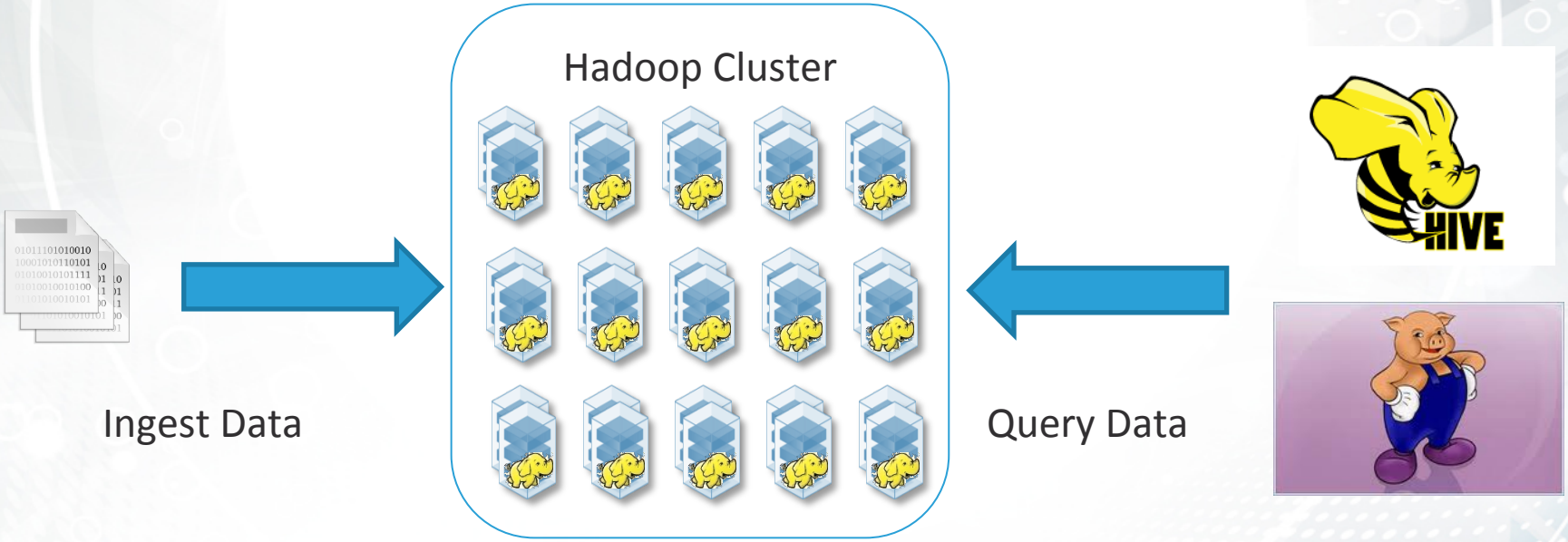

MapReduce needs to keep up with data ingest

hadoop MapReduce

Structured Data!

Query Performance!

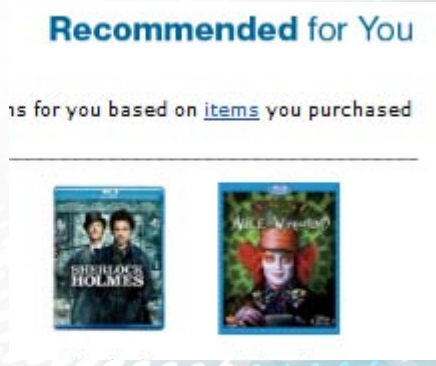
# Leveraging the Results



# Pushing Results further for more Business Analytics

Hadoop Cluster

Data warehouse




hadoop MapReduce

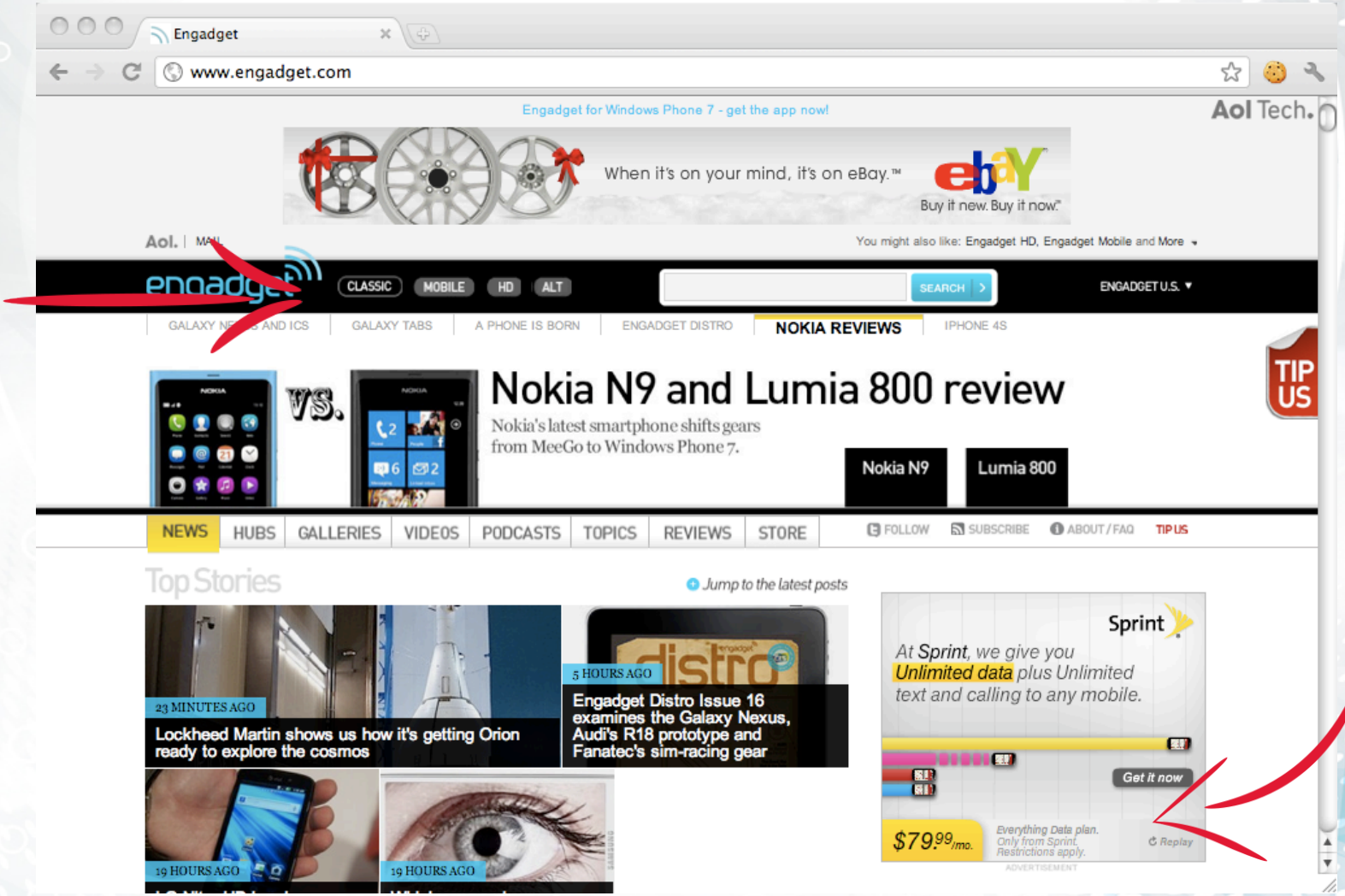
APACHE HBASE

Cassandra

- Keep up with Data Ingest
- External SLAs!
- Data is more valuable if its current!

media6degrees

# Impressions look like…

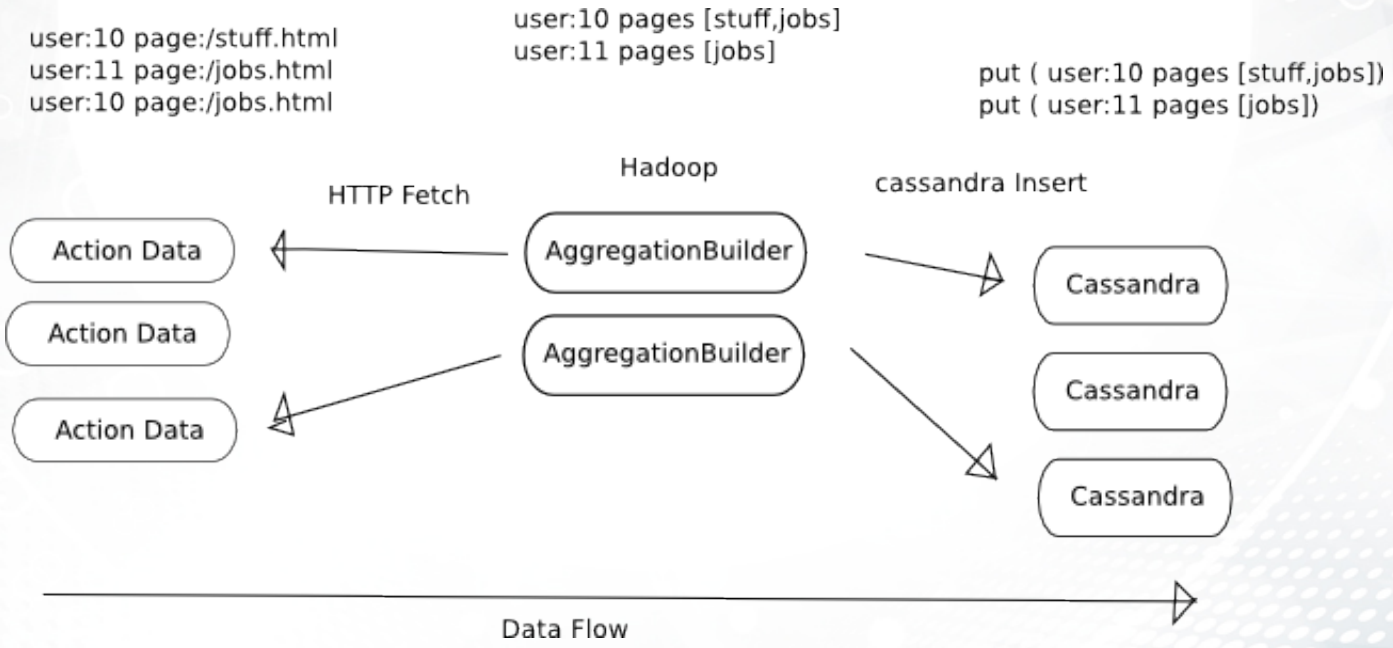


# A High Level look at RTB


1. Browsers visit Publishers and create impressions.
2. Publishers sell impressions via Exchanges.
3. Exchanges serve as auction houses for the impressions
4. On behalf of the marketer, m6d bids the impressions via the auction house. If m6d wins, we display our ad to the browser.

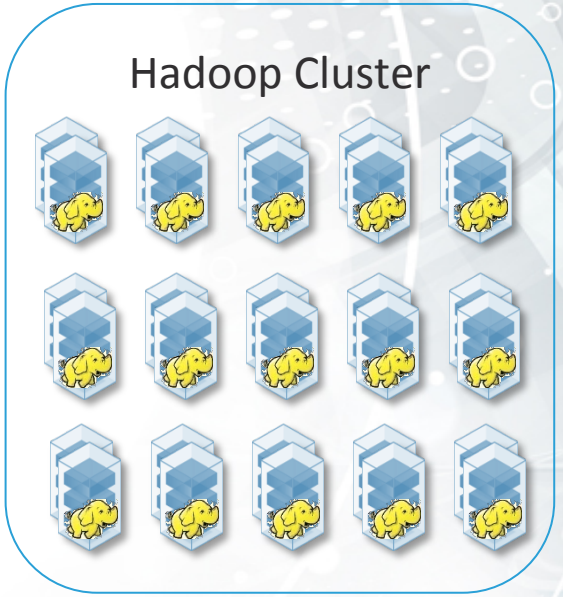

hadoop
MapReduce

# Typical MapReduce Job



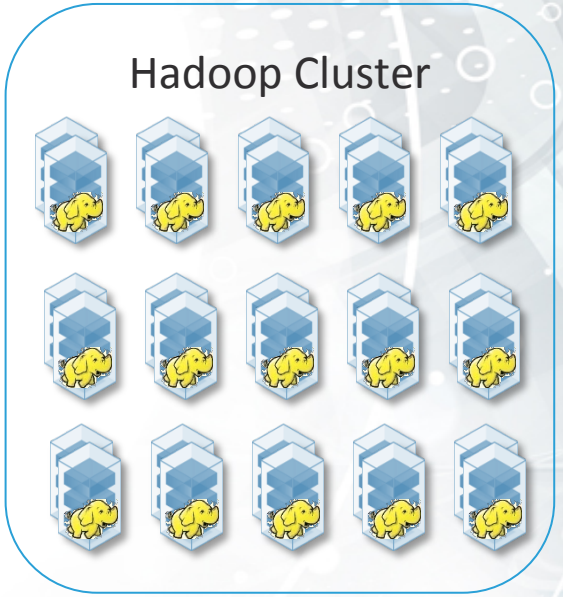
# Hadoop at media6degrees

- Critical piece of infrastructure
- Long Term Data Storage
  - Raw logs
  - Aggregations
  - Reports
  - Generated data (feed back loops)
- Numerous ETL (Extract Transform Load)
- Scheduled and adhoc processes
- Used directly by Tech-Team, Ad Ops, Data Science



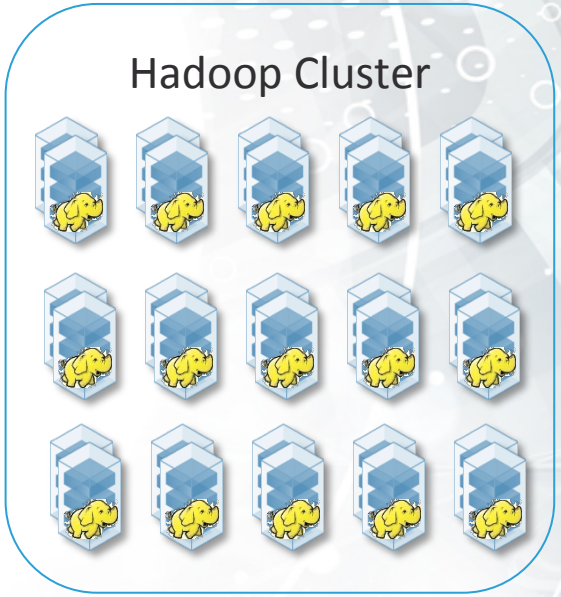
# Hadoop at media6degrees

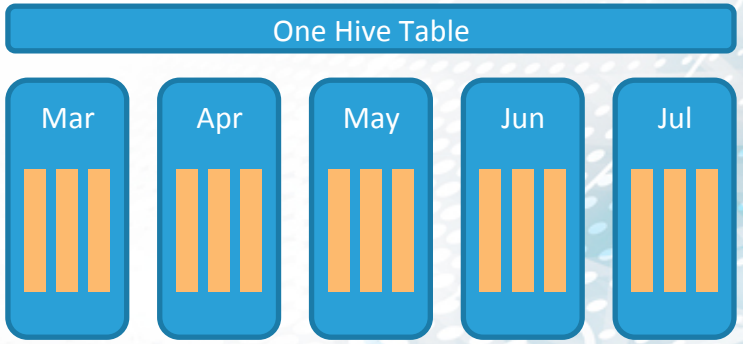
- Two deployments 'production' and 'research'
  - ~ 500 TB - 40+ Nodes
  - ~ 350 TB – 20+ Nodes
- Thousands of jobs
  - <5 minute jobs and 12 hour Job Flows
  - Mostly Hive Jobs
  - Some custom code and streaming jobs



# Most important Design Tenants

- Data Locality
- Partition data for good distribution
  - By time interval
    - Partition pruning with WHERE
  - Clustering (aka bucketing)
    - Optimized sampling and joins
  - Columnar
    - Column oriented



# The Price for Scale out

# The Distribution Challenge

- Data Skew
- Non Uniform Computation (long running tasks)
- Raw Data Growth
- Data features change

Time Consuming

Never be optimal for everything

# Intermediate I/O – Shuffle Bottleneck

- Compression codec
- Block size
- Split-table formats
- Tuning Hadoop variables (spills, buffers, Etc, etc)

Time Consuming

A lot of Trial and Error

Never be optimal for everything

# Optimizing Your Map/Reduce Jobs directly

- Solves Problems at its roots
  - Better performance
  - Less hardware
  - Less data?

Code Changes

A lot of Trial and Error

- Hadoop Custom Job Counters
- Extensive Logging

Log Analysis is the poor man's APM

Very Complicated and Time Consuming

Test Data is not like Real Data

# What about Hive or Pig?

- Solves Problems at its roots
  - Better performance
  - Less hardware
  - Less data?

~~Code Changes~~

A lot of Trial and Error

- ~~Custom Job Counters~~
- ~~Extensive Logging~~

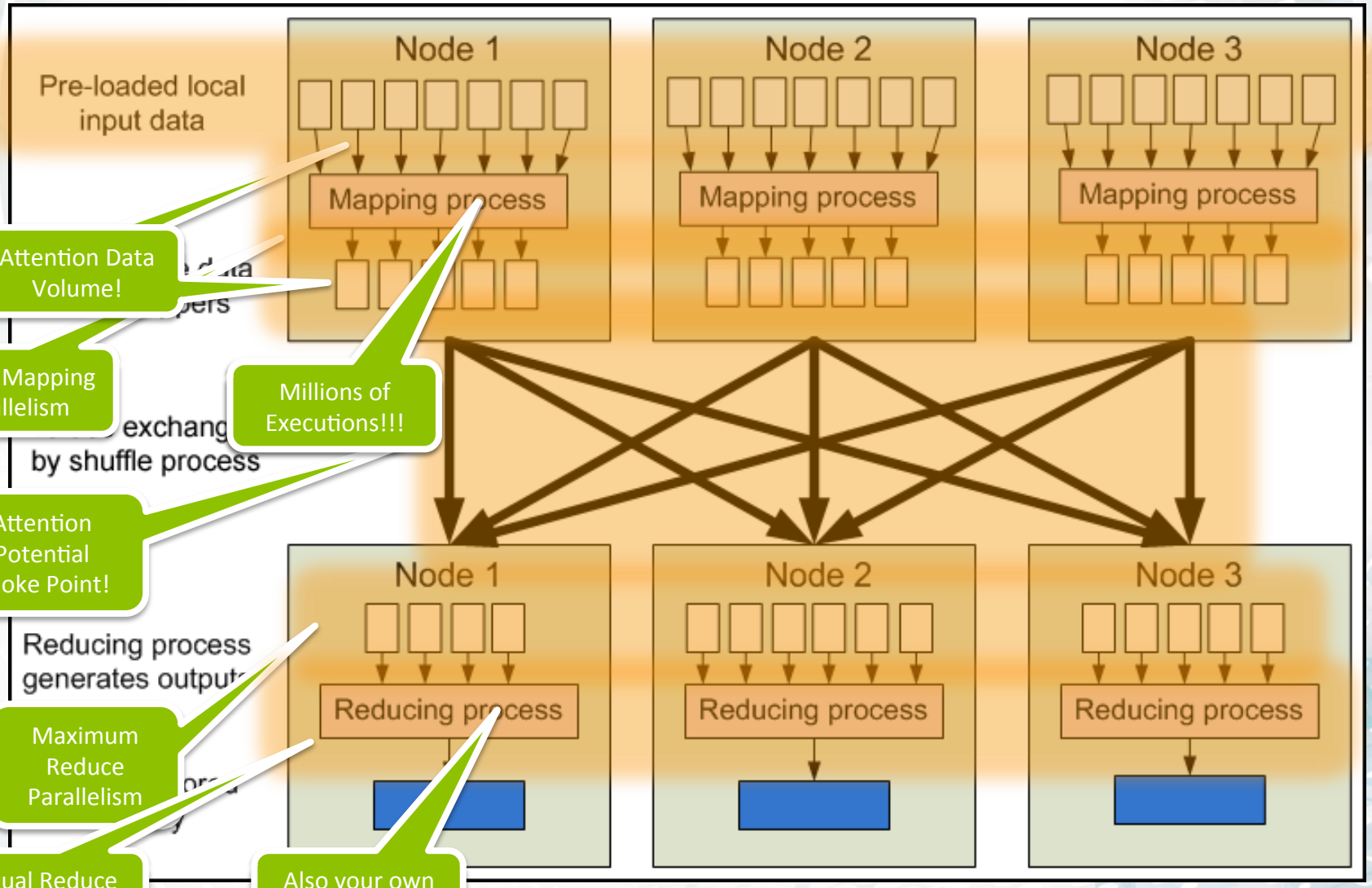
Log Analysis is the poor man's APM

Expert Required

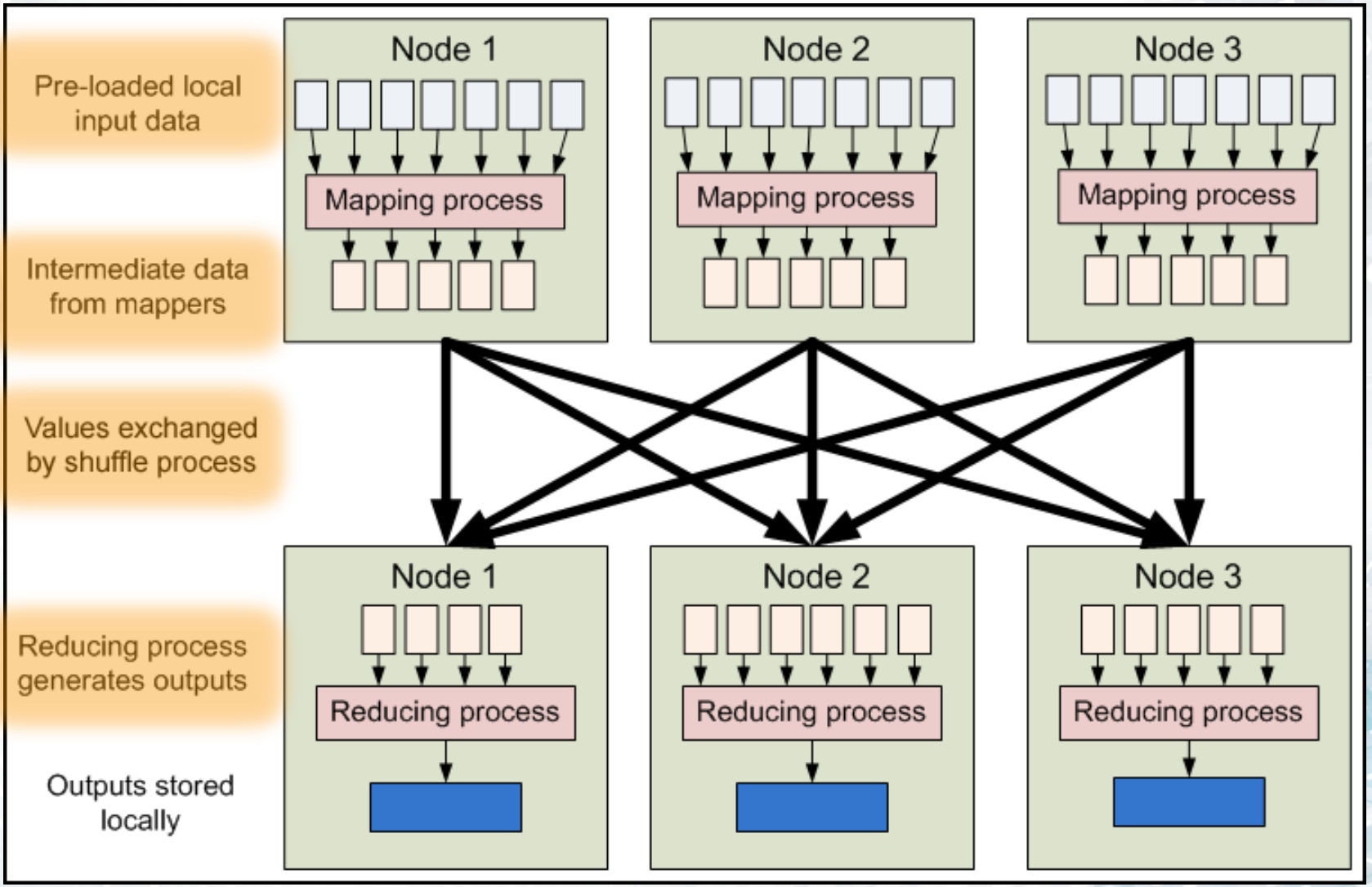
Very Complicated and Time Consuming

# Optimize MapReduce

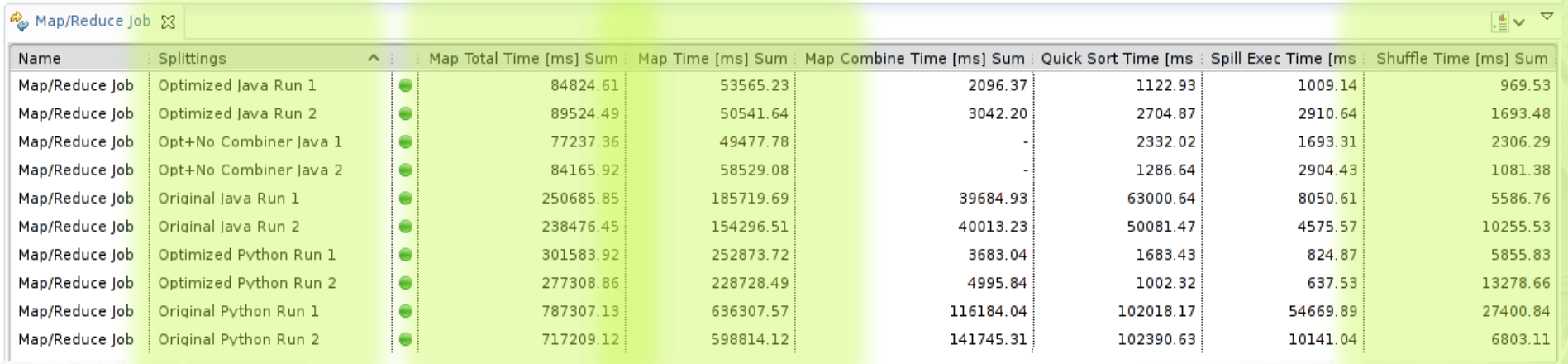
# Understanding Map/Reduce Performance



# Understanding Map/Reduce Performance



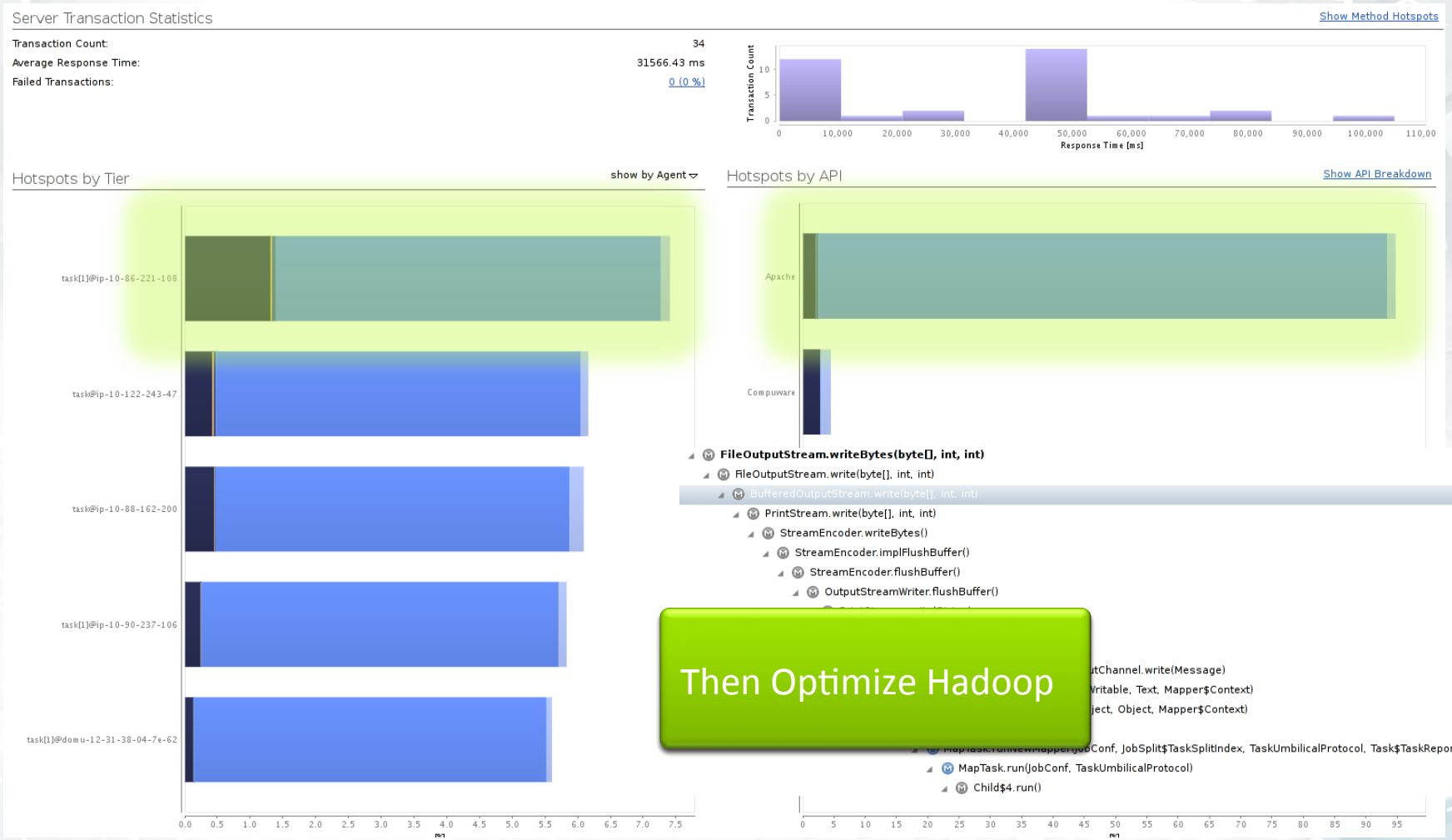
# Map/Reduce Performance Breakdown



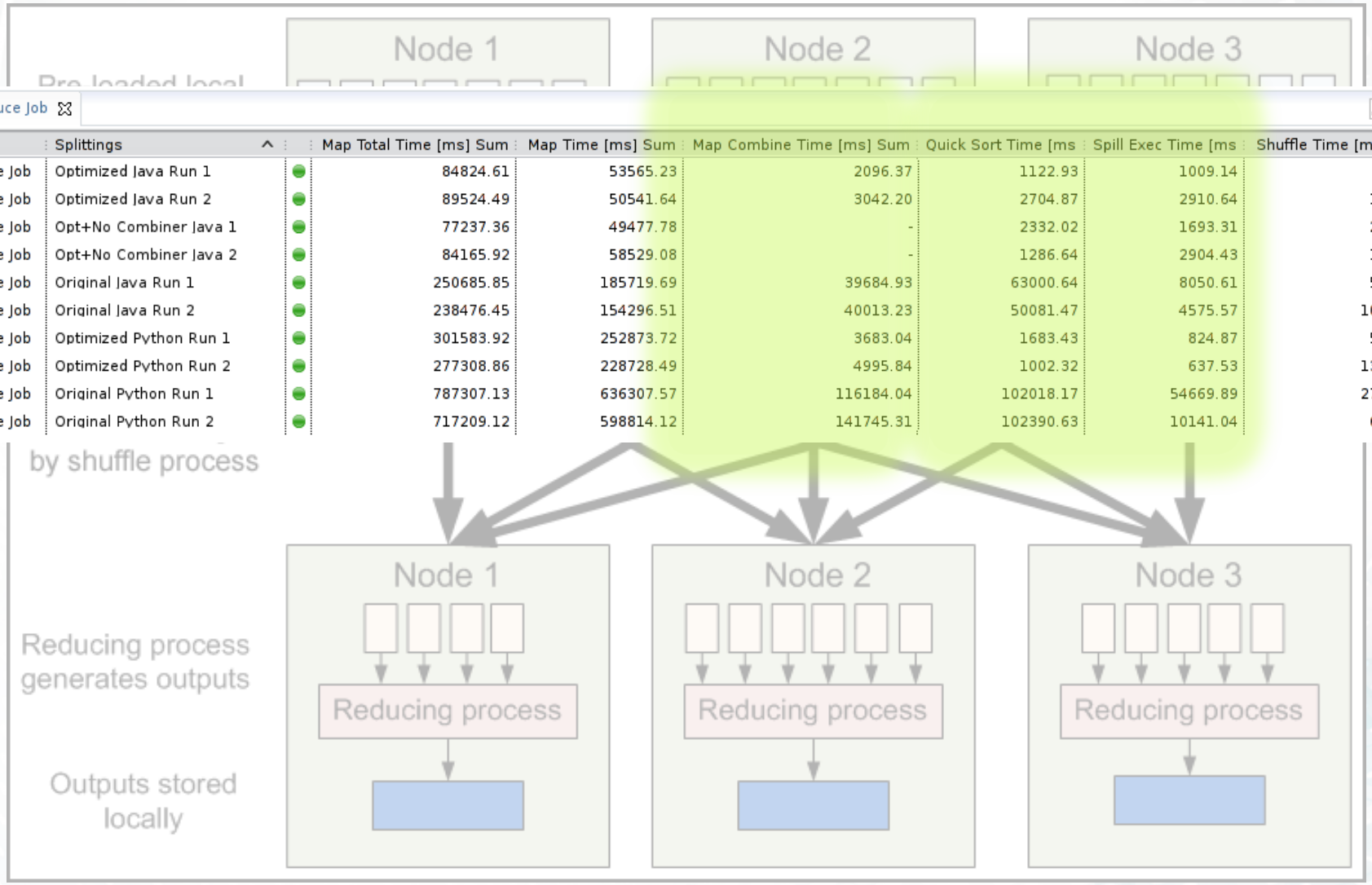
Map/Reduce Job

| Name | Splittings | Map Total Time [ms] Sum | Map Time [ms] Sum | Map Combine Time [ms] Sum | Quick Sort Time [ms | Spill Exec Time [ms | Shuffle Time [ms] Sum |
|---|---|---|---|---|---|---|---|
| Map/Reduce Job | Optimized Java Run 1 | 84824.61 | 53565.23 | 2096.37 | 1122.93 | 1009.14 | 969.53 |
| Map/Reduce Job | Optimized Java Run 2 | 89524.49 | 50541.64 | 3042.20 | 2704.87 | 2910.64 | 1693.48 |
| Map/Reduce Job | Opt+No Combiner Java 1 | 77237.36 | 49477.78 | - | 2332.02 | 1693.31 | 2306.29 |
| Map/Reduce Job | Opt+No Combiner Java 2 | 84165.92 | 58529.08 | - | 1286.64 | 2904.43 | 1081.38 |
| Map/Reduce Job | Original Java Run 1 | 250685.85 | 185719.69 | 39684.93 | 63000.64 | 8050.61 | 5586.76 |
| Map/Reduce Job | Original Java Run 2 | 238476.45 | 154296.51 | 40013.23 | 50081.47 | 4575.57 | 10255.53 |
| Map/Reduce Job | Optimized Python Run 1 | 301583.92 | 252873.72 | 3683.04 | 1683.43 | 824.87 | 5855.83 |
| Map/Reduce Job | Optimized Python Run 2 | 277308.86 | 228728.49 | 4995.84 | 1002.32 | 637.53 | 13278.66 |
| Map/Reduce Job | Original Python Run 1 | 787307.13 | 636307.57 | 116184.04 | 102018.17 | 54669.89 | 27400.84 |
| Map/Reduce Job | Original Python Run 2 | 717209.12 | 598814.12 | 141745.31 | 102390.63 | 10141.04 | 6803.11 |

# Distribution and Hotspots



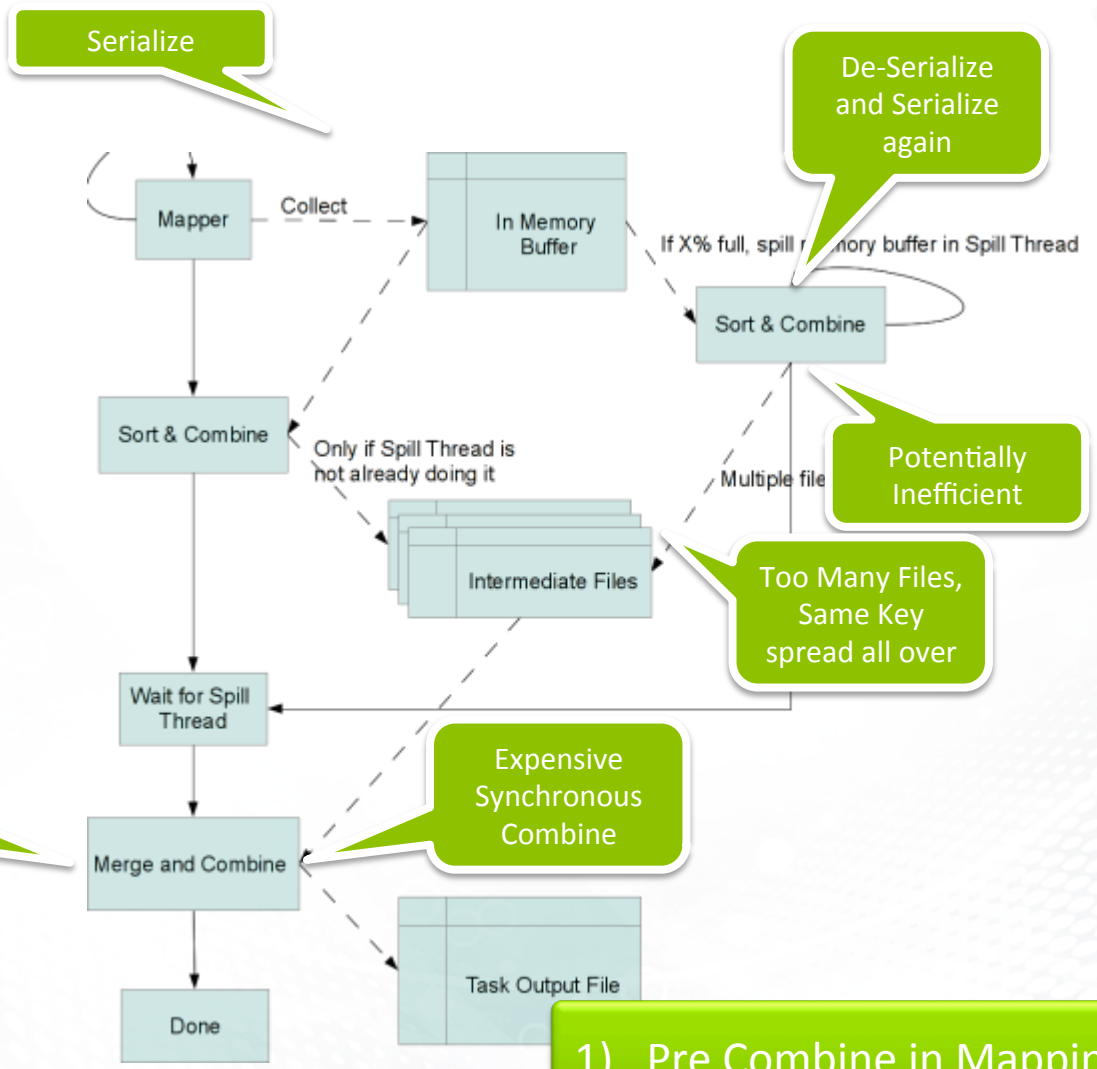
# Combine and Spill Performance




Map/Reduce Job

| Name | Splittings | Map Total Time [ms] Sum | Map Time [ms] Sum | Map Combine Time [ms] Sum | Quick Sort Time [ms | Spill Exec Time [ms | Shuffle Time [ms] Sum |
|---|---|---|---|---|---|---|---|
| Map/Reduce Job | Optimized Java Run 1 | 84824.61 | 53565.23 | 2096.37 | 1122.93 | 1009.14 | 969.53 |
| Map/Reduce Job | Optimized Java Run 2 | 89524.49 | 50541.64 | 3042.20 | 2704.87 | 2910.64 | 1693.48 |
| Map/Reduce Job | Opt+No Combiner Java 1 | 77237.36 | 49477.78 | - | 2332.02 | 1693.31 | 2306.29 |
| Map/Reduce Job | Opt+No Combiner Java 2 | 84165.92 | 58529.08 | - | 1286.64 | 2904.43 | 1081.38 |
| Map/Reduce Job | Original Java Run 1 | 250685.85 | 185719.69 | 39684.93 | 63000.64 | 8050.61 | 5586.76 |
| Map/Reduce Job | Original Java Run 2 | 238476.45 | 154296.51 | 40013.23 | 50081.47 | 4575.57 | 10255.53 |
| Map/Reduce Job | Optimized Python Run 1 | 301583.92 | 252873.72 | 3683.04 | 1683.43 | 824.87 | 5855.83 |
| Map/Reduce Job | Optimized Python Run 2 | 277308.86 | 228728.49 | 4995.84 | 1002.32 | 637.53 | 13278.66 |
| Map/Reduce Job | Original Python Run 1 | 787307.13 | 636307.57 | 116184.04 | 102018.17 | 54669.89 | 27400.84 |
| Map/Reduce Job | Original Python Run 2 | 717209.12 | 598814.12 | 141745.31 | 102390.63 | 10141.04 | 6803.11 |

# Map/Reduce behind the scenes




1) Pre Combine in Mapping Step
2) Avoid many intermediate files and combines

# Identifying Root Cause for failed Jobs

| | | | | | | |
|---|---|---|---|---|---|---|
| Map/Reduce Job | job_201205091829_0005 | 39 | 743401.36 | 220710.25 | | 2.56 % |
| Map/Reduce Job | job_201205090741_0003 | 38 | 234593.32 | 76365.02 | | 2.63 % |
| Map/Reduce Job | job_201205091111_0006 | 38 | 682580.71 | 237163.98 | | 0 % |
| Map/Reduce Job | job_201205090741_0002 | 38 | 235740.73 | 87380.53 | | 0 % |
| Map/Reduce Job | job_201205091829_0003 | 38 | 220933.99 | 72785.27 | | 0 % |
| Map/Reduce Job | job_201205091111_0004 | 38 | 312535.78 | 72077.49 | | 0 % |

Failed Job

| Error Rule | Transactions | Error count | Failing Tran | Failing Page Actions |
|---|---|---|---|---|
| HTTP response codes | 3 | 3 | 0 | 0 |
| HTTP 4xx Response (internal) | 3 | 3 | - | - |

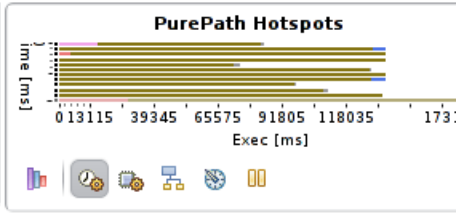
Actual Errors

| Error Rule | Transactions | Cou | HTTP response code | URI |
|---|---|---|---|---|
| HTTP 4xx Response (internal) | 1 | 1 | 404 | https://tmp.dynatrace.s3.amazonaws.com/result%2FJSlow1%2Fpart-r-00000 |
| HTTP 4xx Response (internal) | 1 | 1 | 404 | https://tmp.dynatrace.s3.amazonaws.com/result%2FJSlow1%2Fpart-r-00002 |
| HTTP 4xx Response (internal) | 1 | 1 | 404 | https://tmp.dynatrace.s3.amazonaws.com/result%2FJSlow1%2Fpart-r-00001 |

# Identifying Root Cause for failed Jobs

| PurePath | Response Time [ms | Breakdown | Size | Agent | Application |
|---|---|---|---|---|---|
| ReduceTask.run(JobConf job, TaskUmbilicalProtocol | 134951.77 | wait (73%) io | 191 | task[2]@domu-12-31-3 | Default Applicati 2012 |
| ReduceTask.run(JobConf job, TaskUmbilicalProtocol | 101706.40 | wait (82%) io | 194 | task[2]@domu-12-31-3 | Default Applicati 2012 |
| ReduceTask.run(JobConf job, TaskUmbilicalProtocol | 132191.03 | sync wait (50%) io | 200 | task[2]@domu-12-31-3 | Default Applicati 2012 |

PurePaths | Contributors | Errors

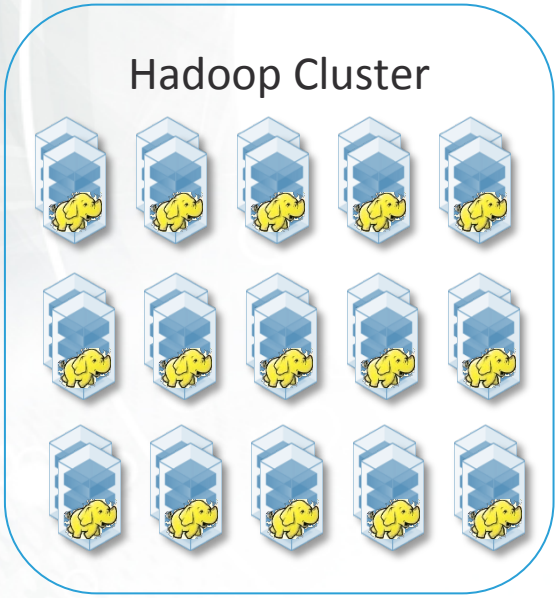



PurePath Tree (showing only relevant nodes) Show all no

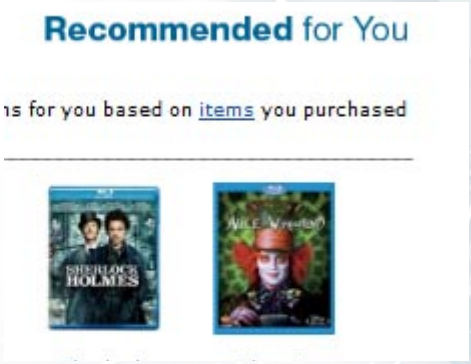
| Method | Argument | Exec Total [ms] | Elapsed Time [ms | Class |
|---|---|---|---|---|
| runNewReducer(JobConf job, TaskUmbilicalProtocol umbilical, Tasks | | 39158.79 | 93001.84 cpu (38%) io (62%) | ReduceTask |
| create(Path f, FsPermission permission, boolean overwrite, int | | 412.80 | 93008.26 io (95%) | NativeS3FileSystem |
| getFileStatus(Path f) | | 386.22 | 93008.28 io (96%) | NativeS3FileSystem |
| getObjectMetadata(String bucketName, String key) | tmp.dynatrace;result/JSlow1/part-r-00000 | 232.78 | 93008.37 io (95%) | AmazonS3Client |
| getObjectMetadata(GetObjectMetadataRequest getObje | | 232.60 | 93008.52 io (96%) | AmazonS3Client |
| executeMethod(HostConfiguration hostconfig, HttpMe | https://tmp.dynatrace.s3.amazonaws.com | 185.47 | 93052.47 io (96%) | HttpClient |
| exception | Not Found | - | 93240.69 | AmazonS3Exception |
| getObjectMetadata(Stri ketName, String key) | tmp.dynatrace;result/JSlow1/part-r-00000 | 67.89 | 93241.19 io (95%) | AmazonS3Client |
| getObjectMetadata(GetObjectMe | | 67.86 | 93241.20 io (95%) | AmazonS3Client |
| executeMethod(HostConfiguration hostconf | | 67.10 | 93241.74 io (96%) | HttpClient |
| exception | | - | 93308.92 | AmazonS3Exception |

Error is in Reducer when writing to S3

# What comes after Hadoop?









# and Application Performance

# Cassandra at m6d for Real Time Bidding

- RTB limited data is provided from exchange
- System to store information on users
  - Frequency Capping
  - Visit History
  - Segments (product service affinity)
- Low latency Requirements
  - Less then 100ms
  - Requires fast read/write on discrete data

# Key Cassandra Design Tenants

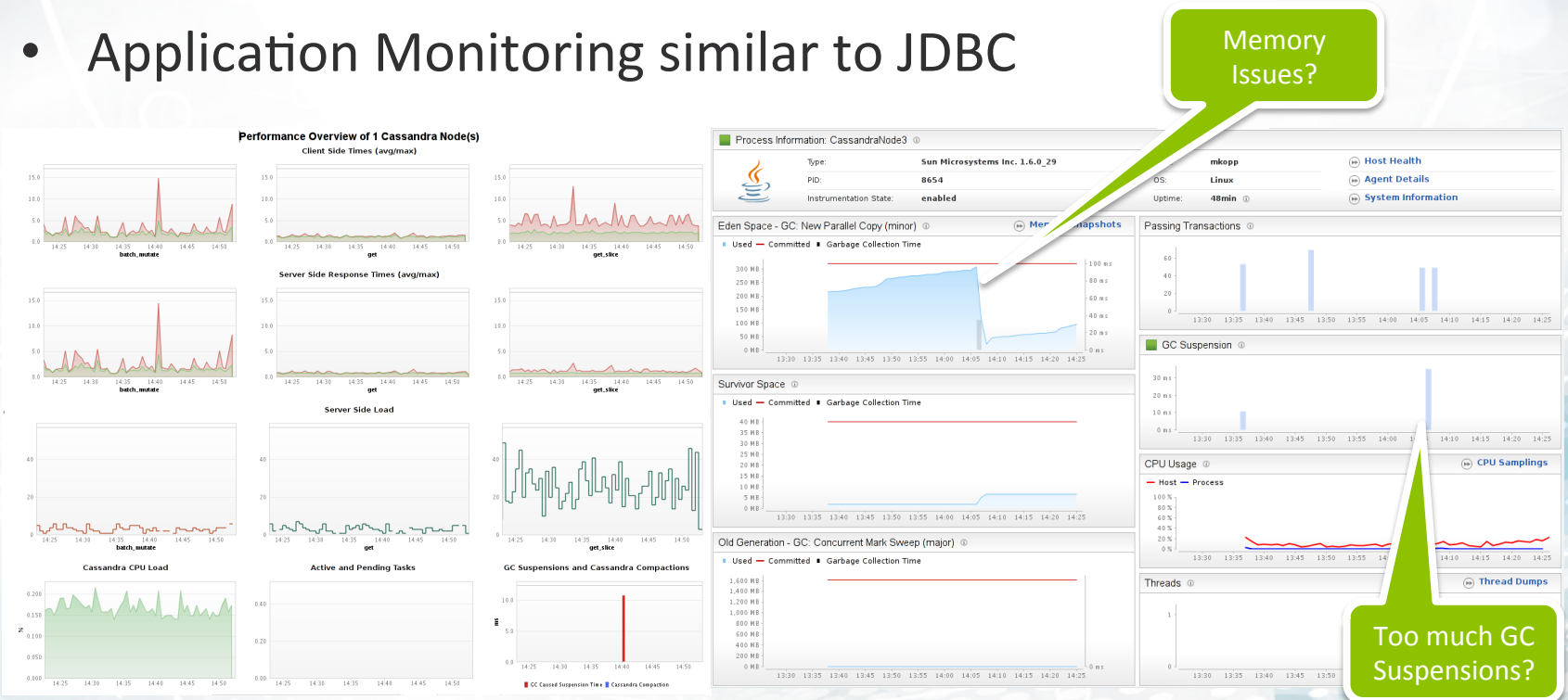
- Swap/paging not possible
- Mostly schema-less
- Writes do not read
  - Read/Write is an anti-pattern
- Optimize around put and get
  - Not for scan and query
- De-Normalize data
  - Attempt to get all data in single read*

# Cassandra Design Challenges

- De-normalize
  - Store data to optimize reads
  - Composite (multi-column) keys
- Multi-column family and Multi-tenant scenarios
- Compress settings
  - Disk and cache savings
  - CPU and JVM costs
- Data/Compaction settings
  - Size tiered vs. LevelDB
- Caching, Memtable and other tuning

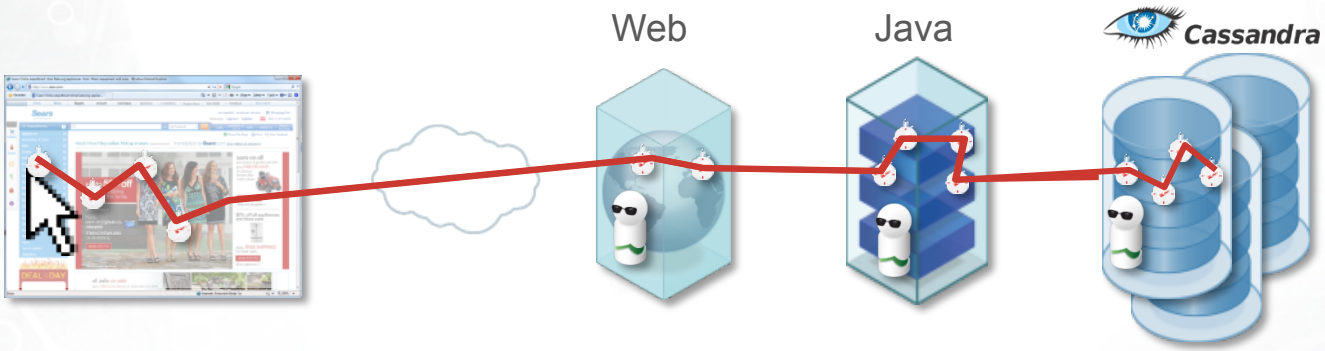
# How to handle performance issues?

- System Monitoring
  - CPU, Disk and Process Health
  - Track req/sec
  - Track size of Column Families, rows and columns (JMX)
- Application Monitoring similar to JDBC




# APM for Cassandra Applications


# NoSQL APM is not so different after all…

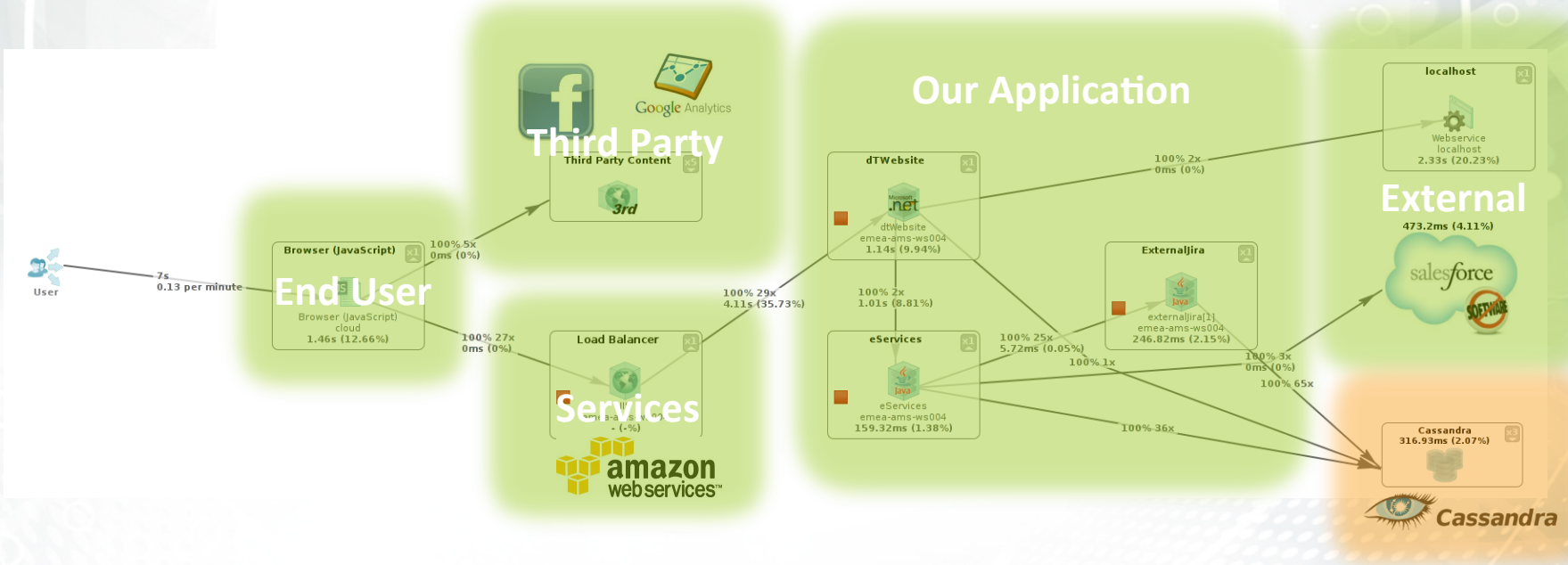



**Key APM Problems Identified**

1) Response Time Contribution
2) data access patterns
3) transaction to query relationship (transaction flow)

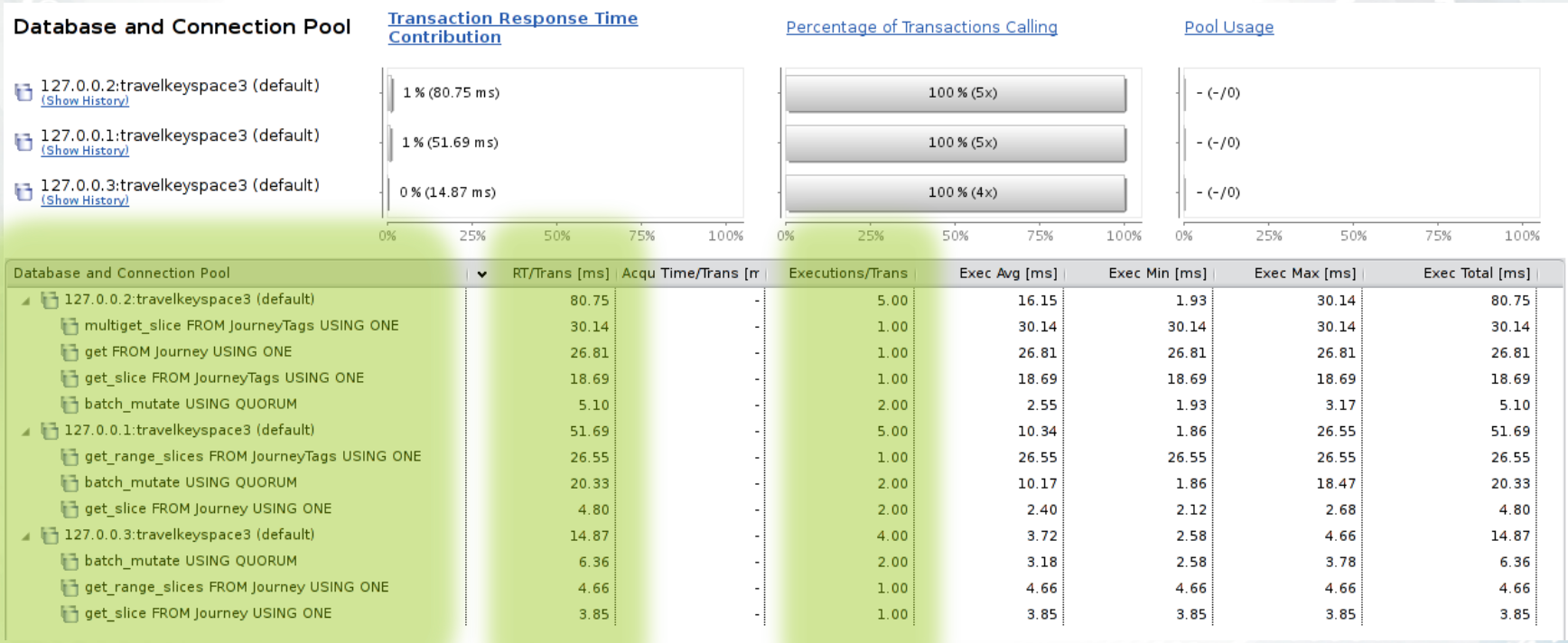
1) Data Access Distribution
2) End-to-End Monitoring
3) Storage (I/O, GC) Bottlenecks
4) Consistency Level

# Real End-to-End Application Performance




End User Response Time Contribution

# Understanding Cassandra's Contribution

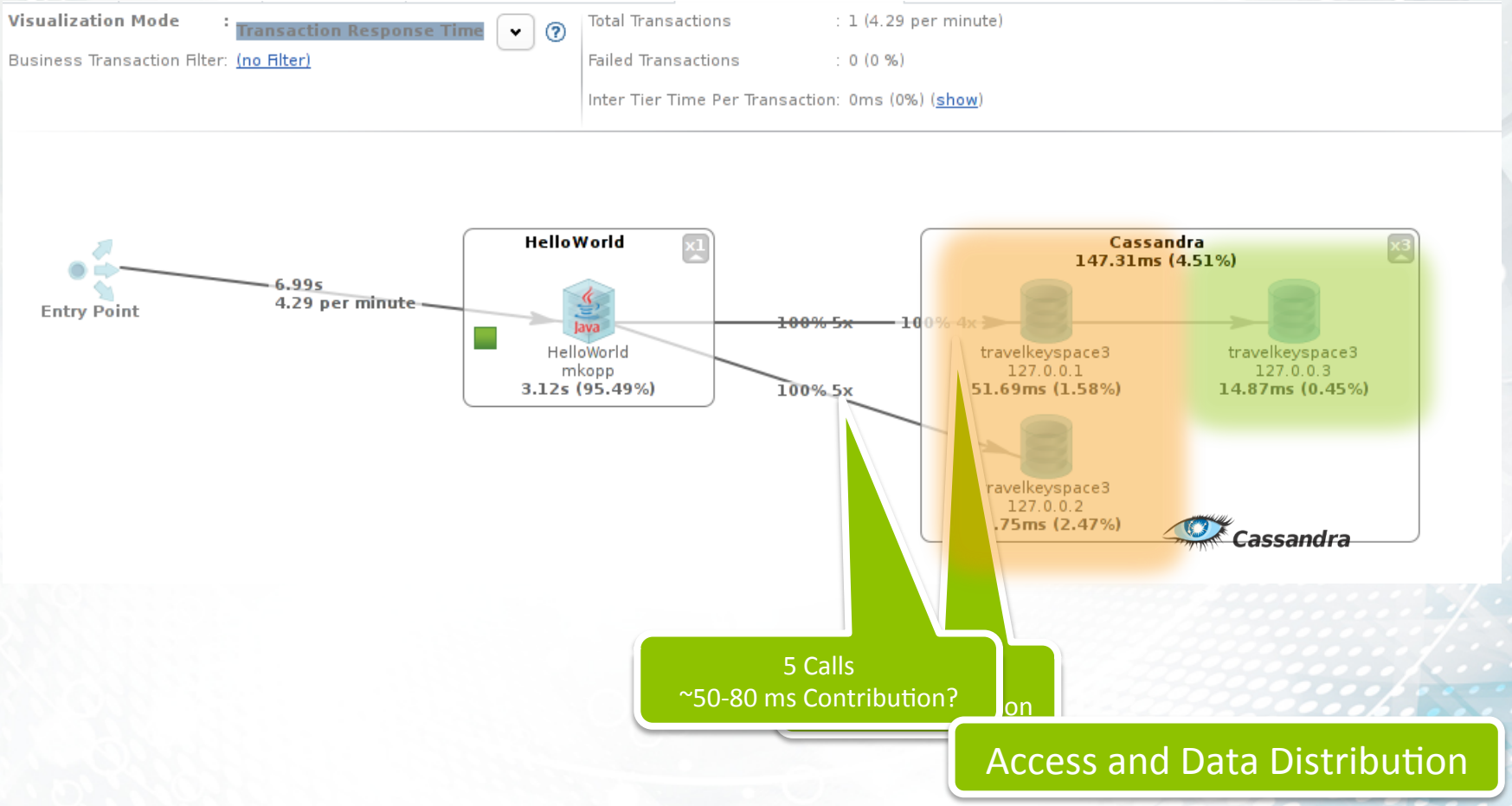



| Database and Connection Pool | RT/Trans [ms] | Acqu Time/Trans [m | Executions/Trans | Exec Avg [ms] | Exec Min [ms] | Exec Max [ms] | Exec Total [ms] |
|---|---|---|---|---|---|---|---|
| 127.0.0.2:travelkeyspace3 (default) | 80.75 | - | 5.00 | 16.15 | 1.93 | 30.14 | 80.75 |
| multiget_slice FROM JourneyTags USING ONE | 30.14 | - | 1.00 | 30.14 | 30.14 | 30.14 | 30.14 |
| get FROM Journey USING ONE | 26.81 | - | 1.00 | 26.81 | 26.81 | 26.81 | 26.81 |
| get_slice FROM JourneyTags USING ONE | 18.69 | - | 1.00 | 18.69 | 18.69 | 18.69 | 18.69 |
| batch_mutate USING QUORUM | 5.10 | - | 2.00 | 2.55 | 1.93 | 3.17 | 5.10 |
| 127.0.0.1:travelkeyspace3 (default) | 51.69 | - | 5.00 | 10.34 | 1.86 | 26.55 | 51.69 |
| get_range_slices FROM JourneyTags USING ONE | 26.55 | - | 1.00 | 26.55 | 26.55 | 26.55 | 26.55 |
| batch_mutate USING QUORUM | 20.33 | - | 2.00 | 10.17 | 1.86 | 18.47 | 20.33 |
| get_slice FROM Journey USING ONE | 4.80 | - | 2.00 | 2.40 | 2.12 | 2.68 | 4.80 |
| 127.0.0.3:travelkeyspace3 (default) | 14.87 | - | 4.00 | 3.72 | 2.58 | 4.66 | 14.87 |
| batch_mutate USING QUORUM | 6.36 | - | 2.00 | 3.18 | 2.58 | 3.78 | 6.36 |
| get_range_slices FROM Journey USING ONE | 4.66 | - | 1.00 | 4.66 | 4.66 | 4.66 | 4.66 |
| get_slice FROM Journey USING ONE | 3.85 | - | 1.00 | 3.85 | 3.85 | 3.85 | 3.85 |

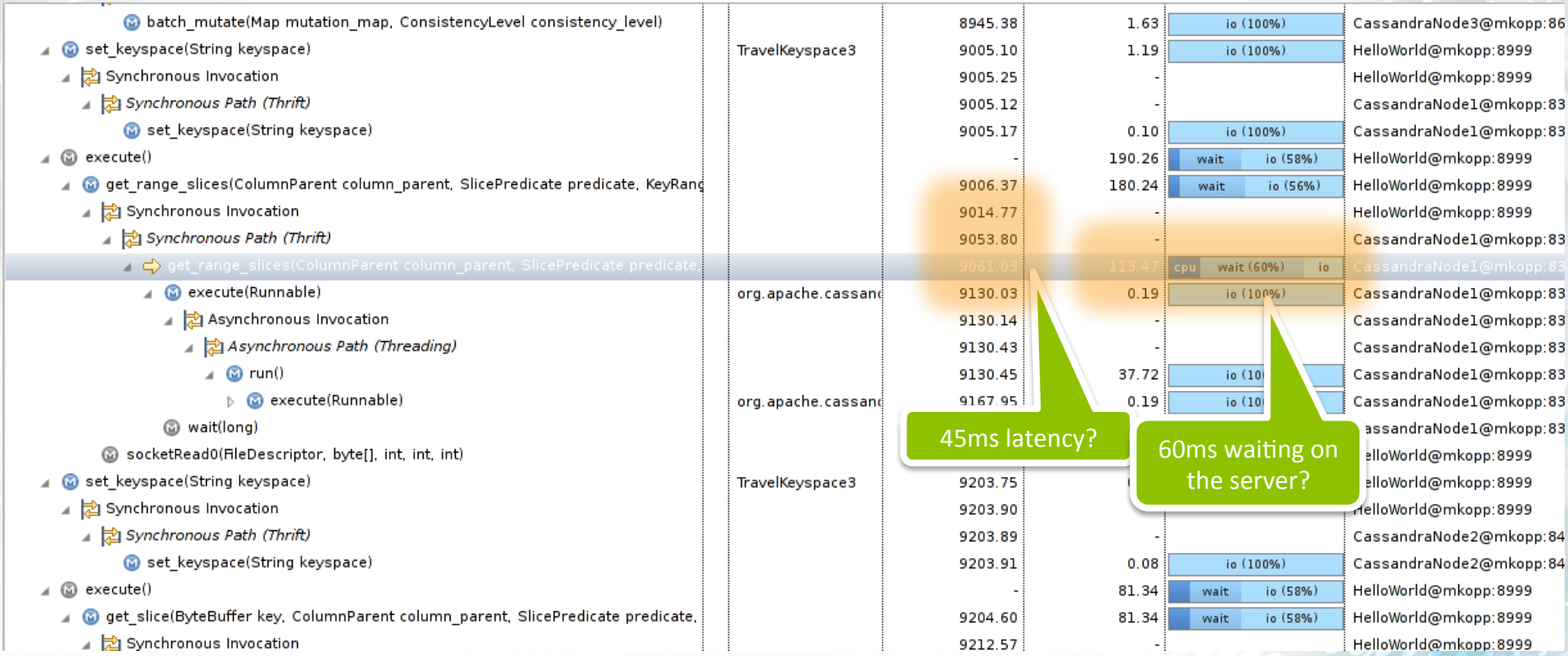
Which statements did the Transaction Execute?
Which node
Contribution of
Too many calls?
Which Consistency Level was used?

Data Access patterns

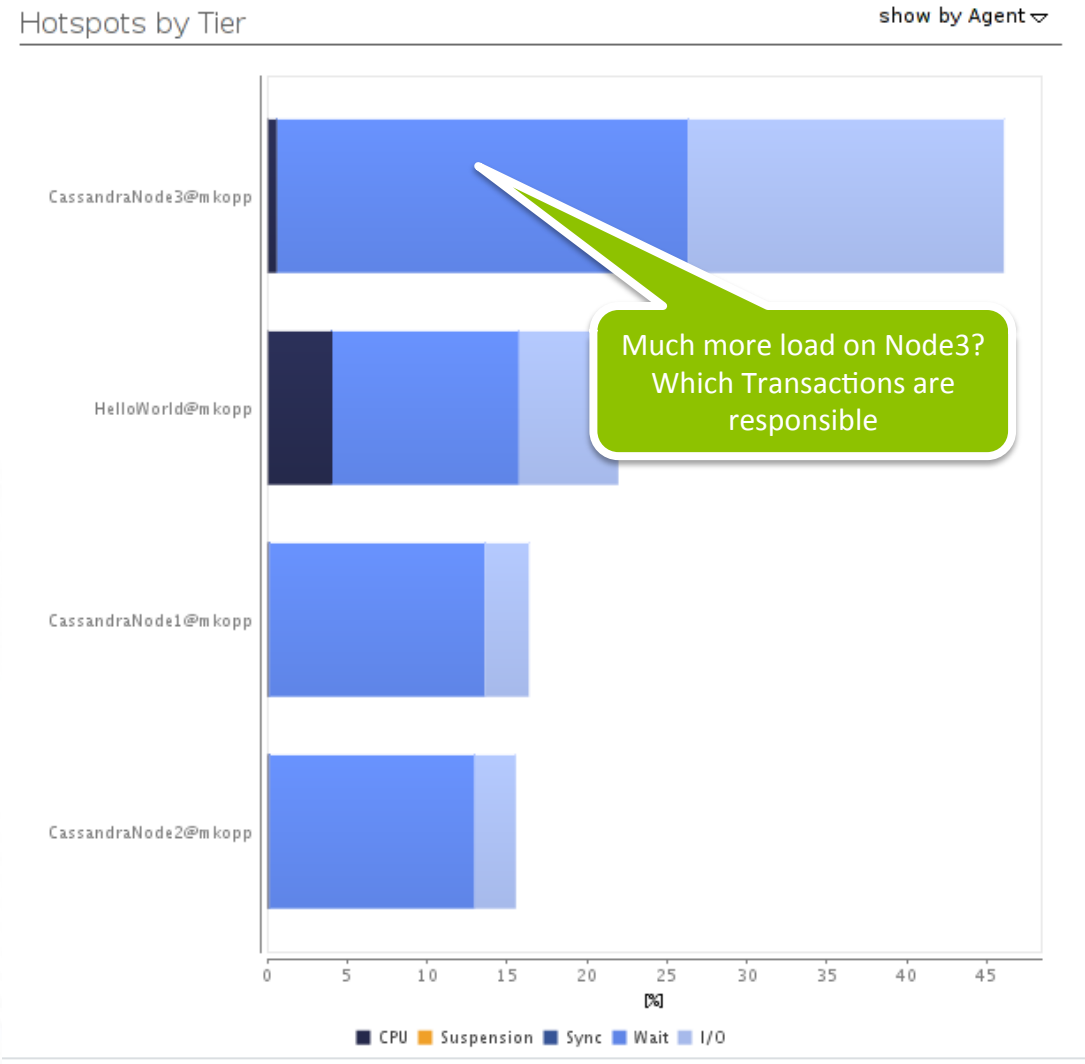
# Understand Response Time Contribution



# Why and how was a statement executed?



# Any Hotspots on the Cassandra Nodes?

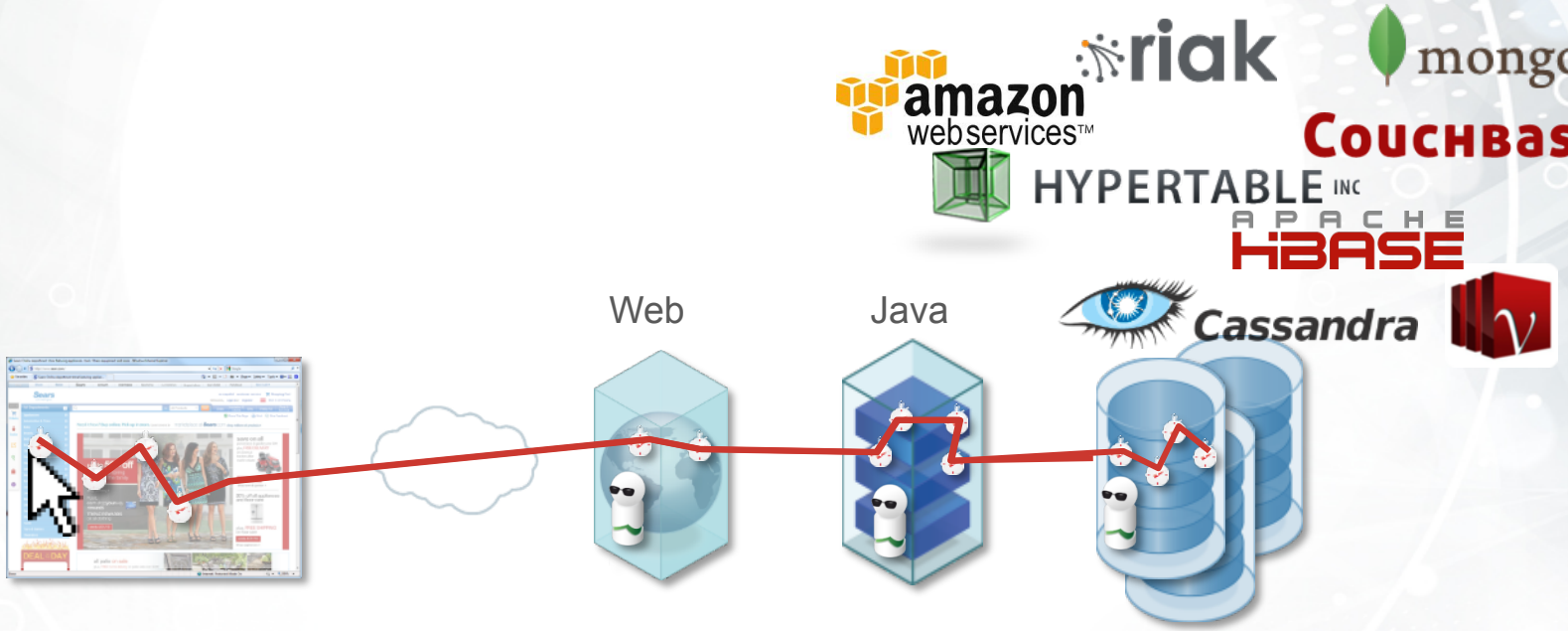


COMPUWARE APM

DYNATRACE® | GOMEZ®

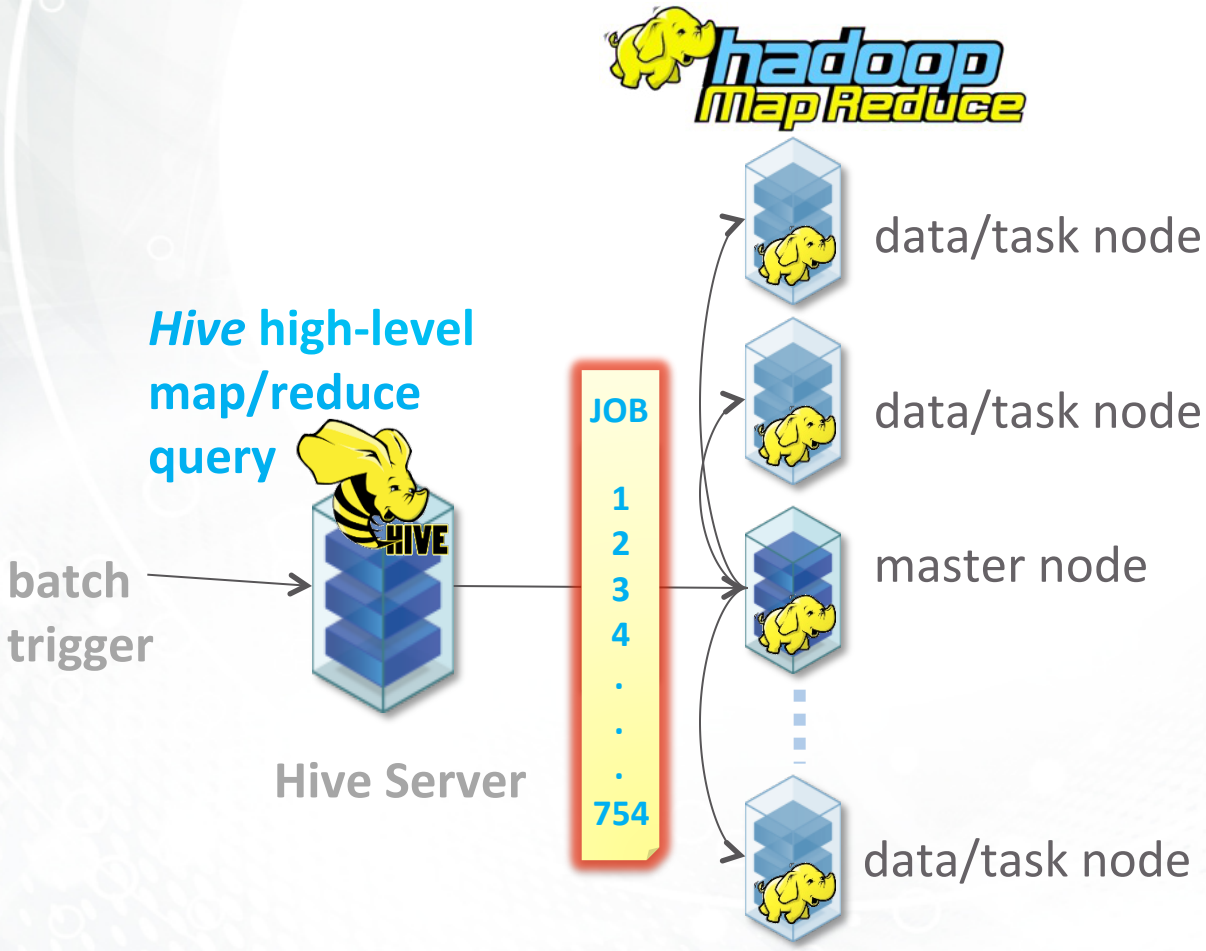
# Conclusion

# Extend Performance Focus on Application




# A Fast Database doesn't make a fast Application

# Intelligent MapReduce APM




# Simple Optimizations with big impact


Q&A


**Michael Kopp**, Technology Strategist

**michael.kopp@compuware.com**

@mikopp

blog.dynatrace.com

book.dynatrace.com

# Monitor MapReduce and Hadoop