



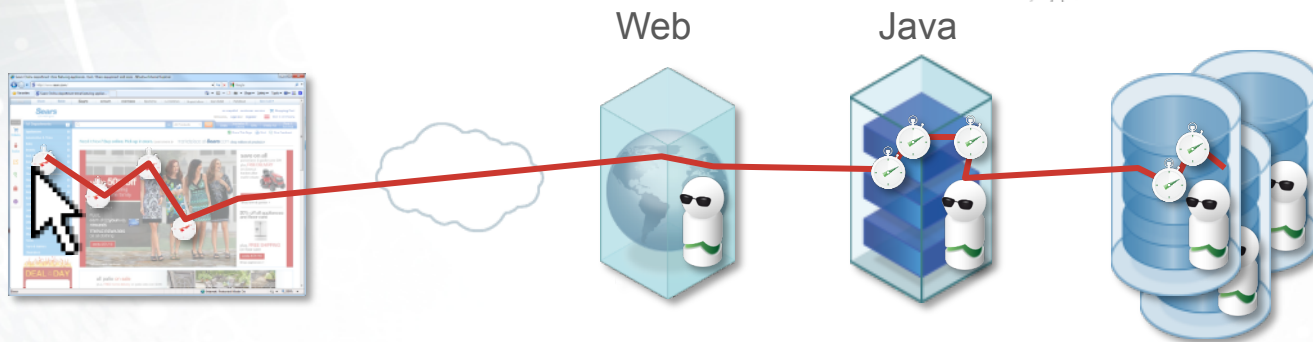
COMPUWARE APM
DYNATRACE® | GOMEZ®

Performance Management in 'Big Data' Applications

Michael Kopp, Technology Strategist

@mikopp

NoSQL: High Volume/Low Latency DBs



Key Challenges

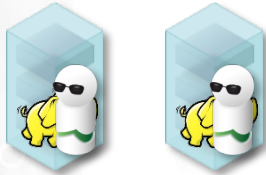
- 1) Even Distribution
- 2) Correct Schema and Access patterns
- 3) Understanding Application Impact
- 4) Proprietary API

Key Benefits

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

Hadoop: Large Scale Parallel Processing

Master Node



Job Tracker
Name Node

Hadoop Cluster



1 Data Node per Host
1 Task Tracker per Host
Many Task JVMs per Host

- 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

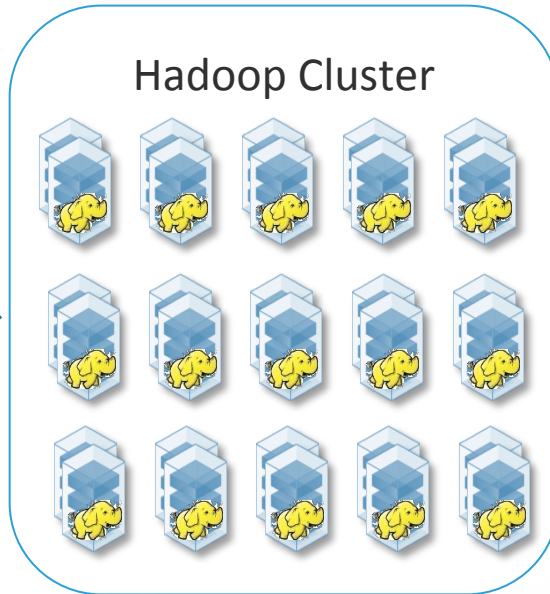
Key Benefits

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

Raw Data ingest



Ingest Data



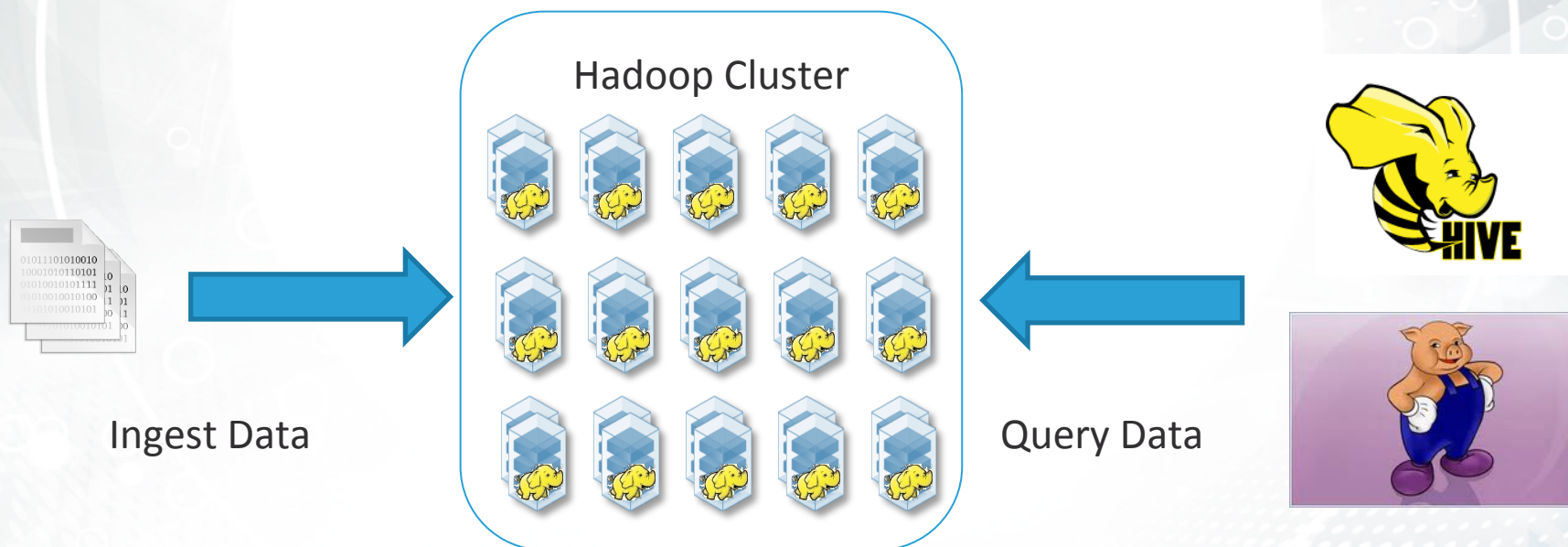
Query Data



Raw Data!

Slow and
Cumbersome

Transform to Hive/Pig Structured Data



MapReduce needs to keep up with data ingest



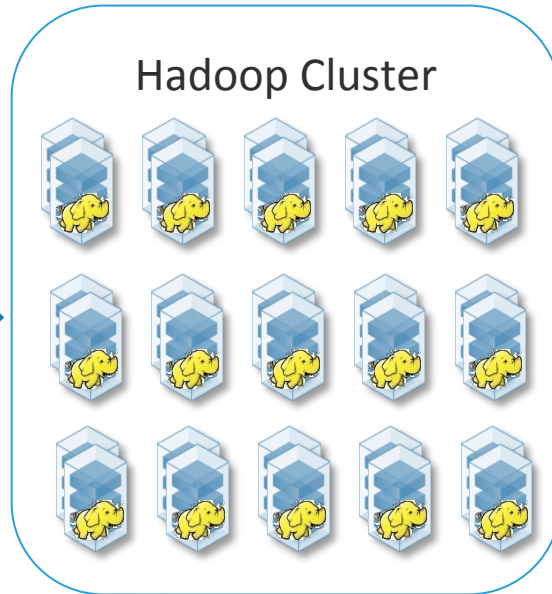
Structured Data!

Query Performance!

Leveraging the Results



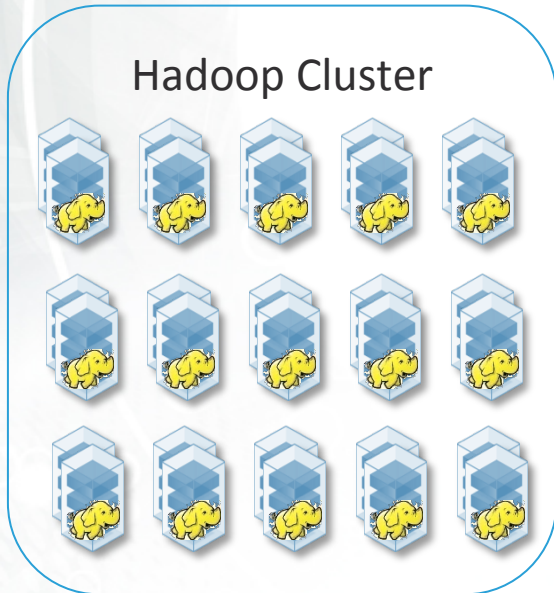
Ingest Data



Query Data



Pushing Results further for more Business Analytics



Data warehouse



APACHE
HBASE

Cassandra



Recommended for You

15 for you based on [items](#) you purchased



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

media6degrees

prospect targeting led by research.

People with similar interests tend to **cluster** into groups with shared brand affinities.

The people clustering with your brand's current customers are your best **prospects**.

m6d identifies the clusters with affinity for your brand.



Impressions look like...

The screenshot shows the Engadget website interface. At the top, there's a navigation bar with the Engadget logo and menu options like CLASSIC, MOBILE, HD, and ALT. A red arrow points to the Engadget logo. Below the navigation bar, there's a main content area featuring a large article titled "Nokia N9 and Lumia 800 review". To the right of this article, there's a "TIP US" button. Below the main article, there's a "Top Stories" section with several article thumbnails. One thumbnail is for "Engadget Distro Issue 16" and another is for "Lockheed Martin shows us how it's getting Orion ready to explore the cosmos". To the right of the "Top Stories" section, there's a large advertisement for Sprint, featuring the text "At Sprint, we give you Unlimited data plus Unlimited text and calling to any mobile." and a "Get it now" button. A red arrow points to the "Get it now" button. The website also features a search bar and a "SEARCH" button. At the bottom, there's a footer with social media links and a "COMPUWARE APM" logo.

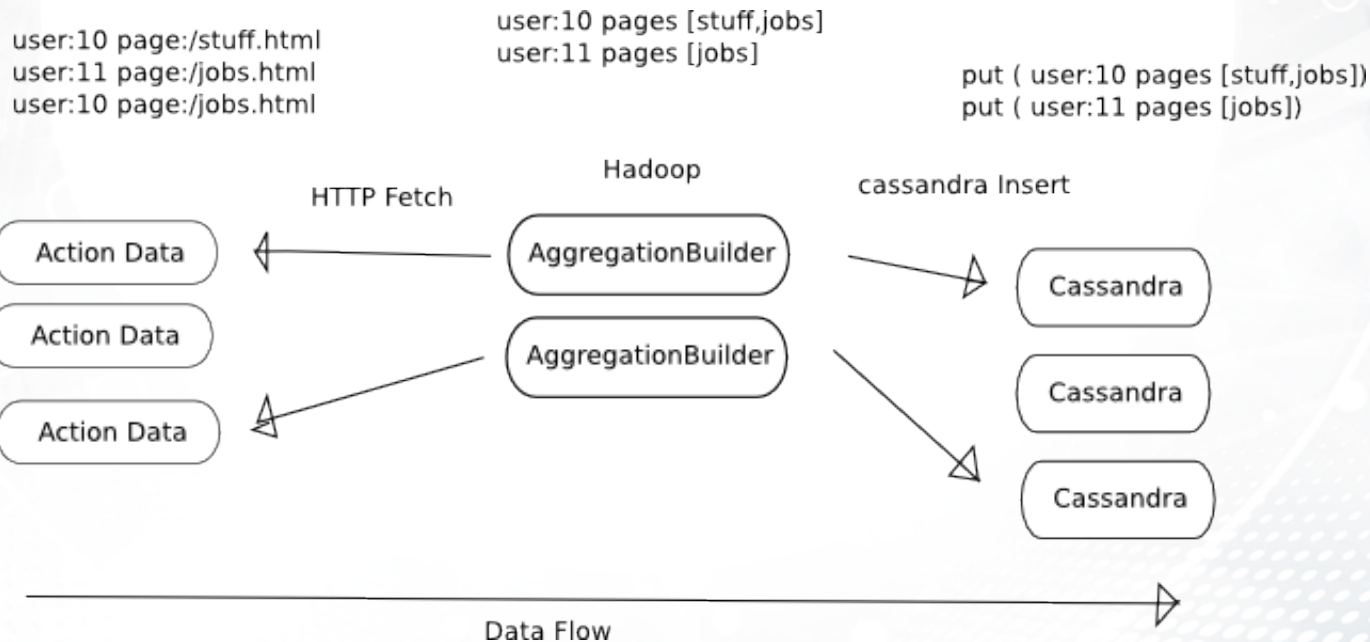
A High Level look at RTB



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.

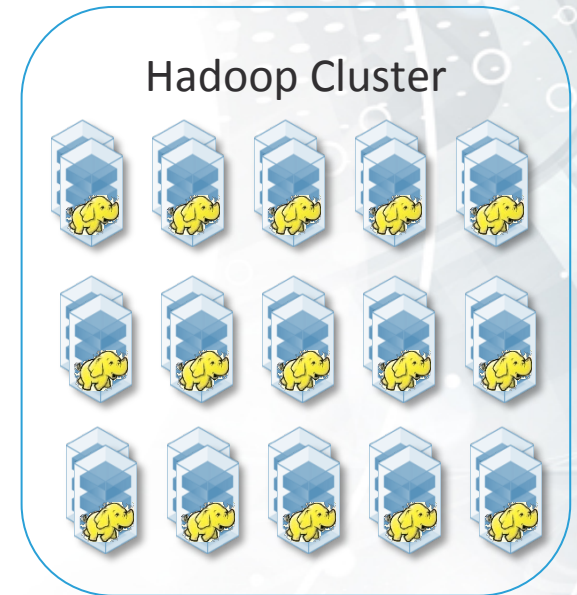


Typical MapReduce Job



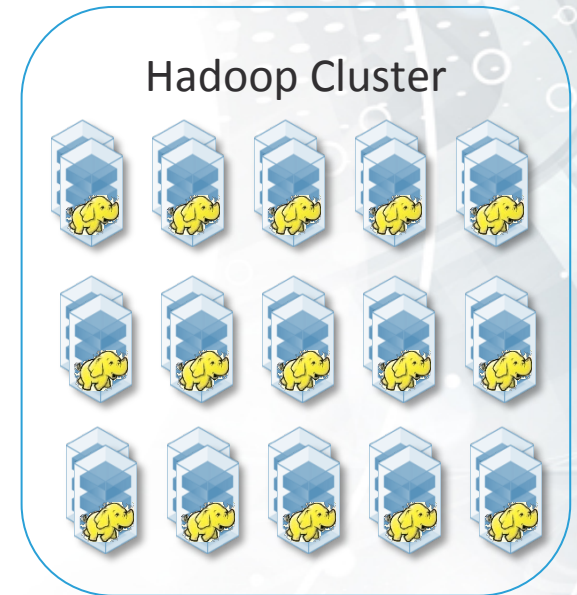
Hadoop at media6degrees

- 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



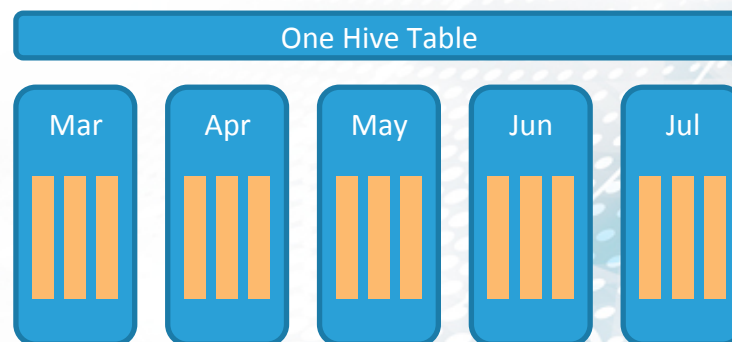
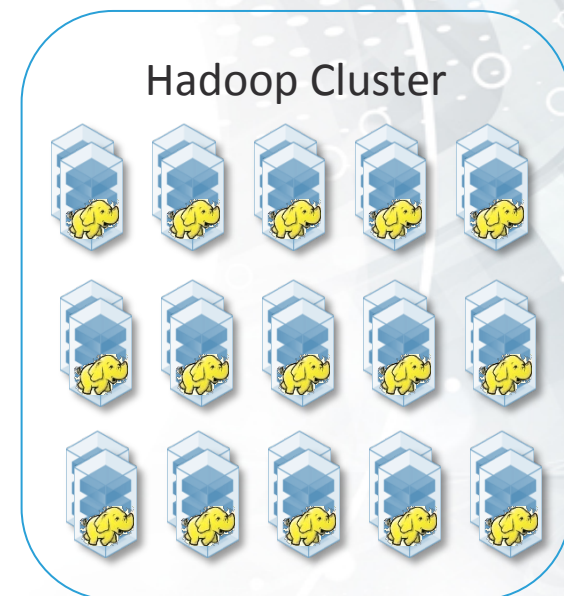
Hadoop at media6degrees

- 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

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```



```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```

```
01011101010010  
100010101110101  
01010010101111  
01010010010100  
01101010010101
```


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?
- Hadoop Custom Job Counters
- Extensive Logging

Test Data is not like
Real Data

**Code
Changes**

**A lot of Trial and
Error**

**Log Analysis is
the poor man's
APM**

**Very
Complicated and
Time Consuming**

What about Hive or Pig?

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

- Custom Job Counters
- Extensive Logging

**Expert
Required**

~~Code
Changes~~

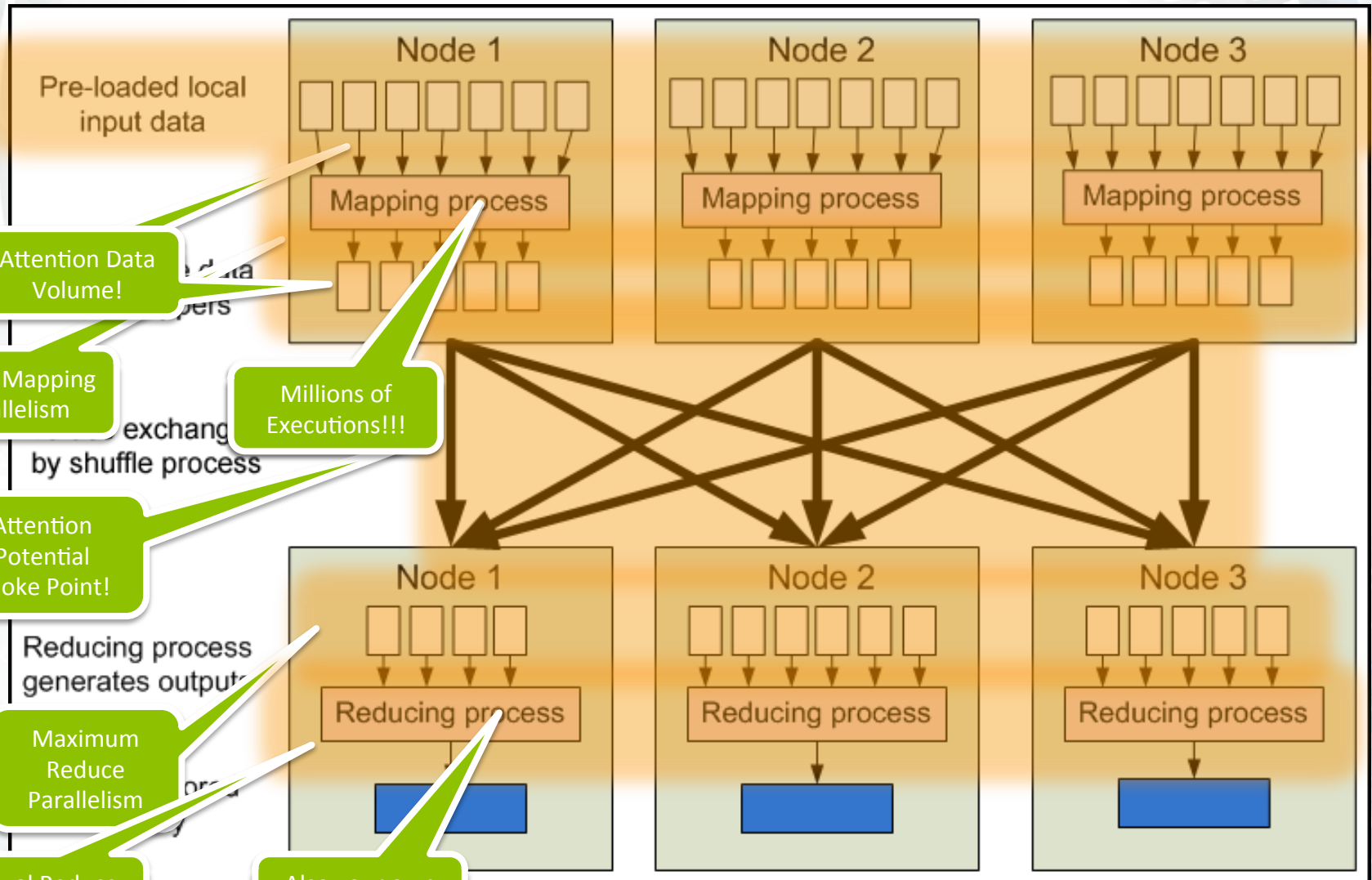
**A lot of Trial and
Error**

**Log Analysis is
the poor man's
APM**

**Very
Complicated and
Time Consuming**

Optimize MapReduce

Understanding Map/Reduce Performance



Attention Data Volume!

Actual Mapping Parallelism

Millions of Executions!!!

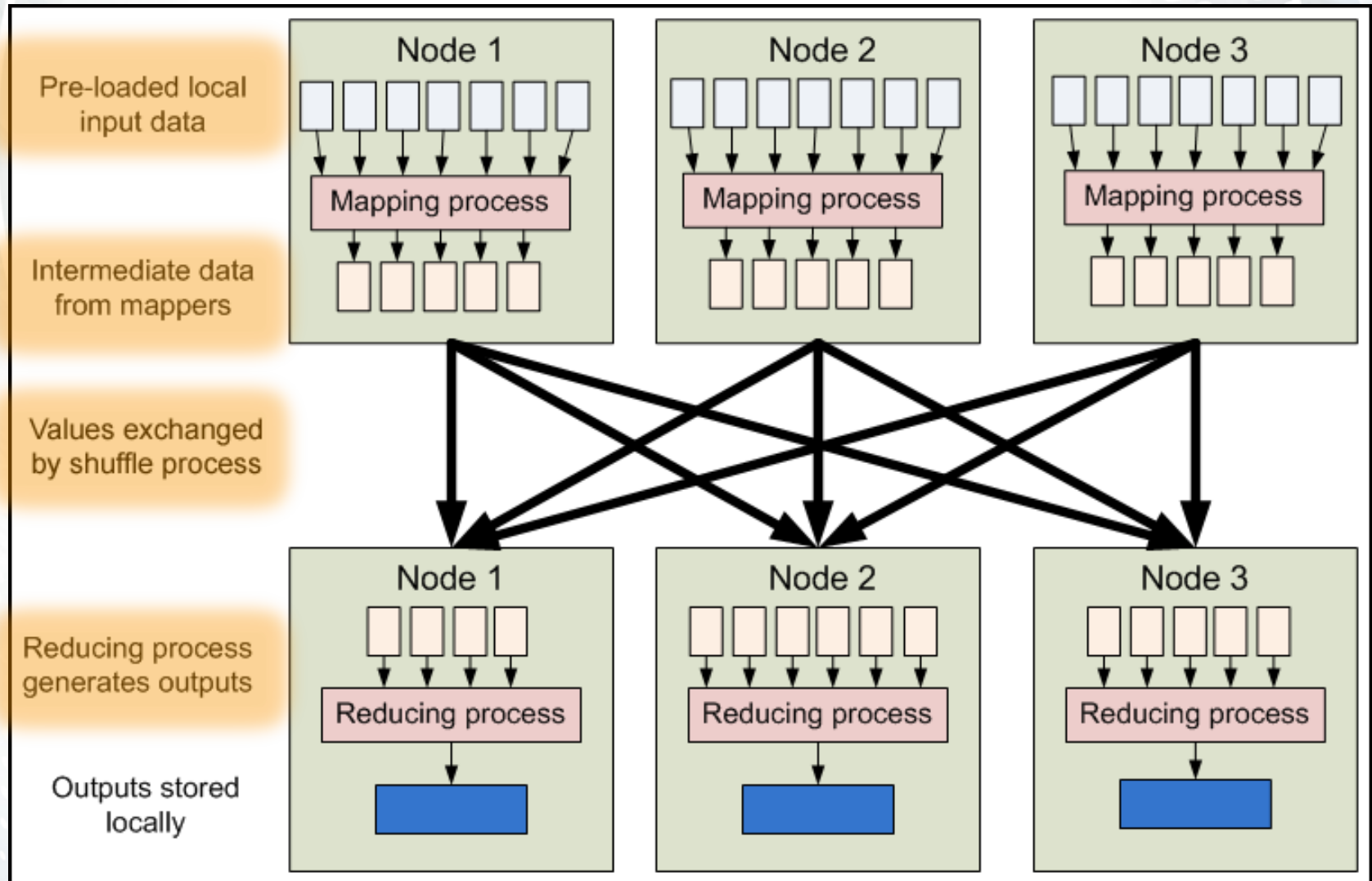
Attention Potential Choke Point!

Maximum Reduce Parallelism

Actual Reduce Parallelism

Also your own Code

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

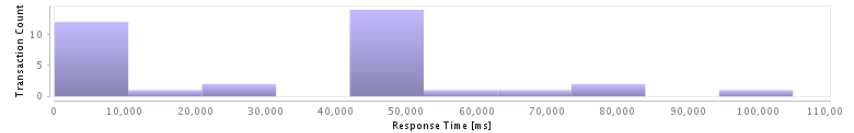
Server Transaction Statistics

[Show Method Hotspots](#)

Transaction Count: 34

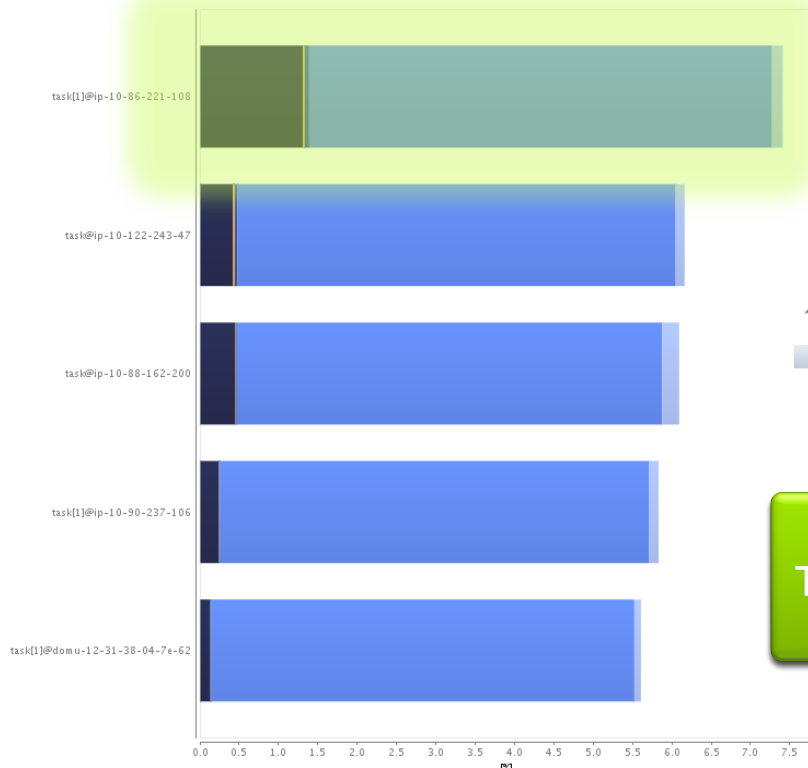
Average Response Time: 31566.43 ms

Failed Transactions: 0 (0%)



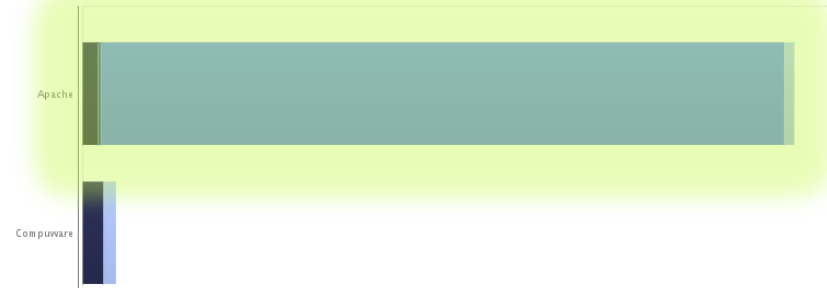
Hotspots by Tier

show by Agent ▾



Hotspots by API

[Show API Breakdown](#)



- FileOutputStream.writeBytes(byte[], int, int)
- FileOutputStream.write(byte[], int, int)
- BufferedOutputStream.write(byte[], int, int)
- PrintStream.write(byte[], int, int)
- StreamEncoder.writeBytes()
- StreamEncoder.implFlushBuffer()
- StreamEncoder.flushBuffer()
- OutputStreamWriter.flushBuffer()

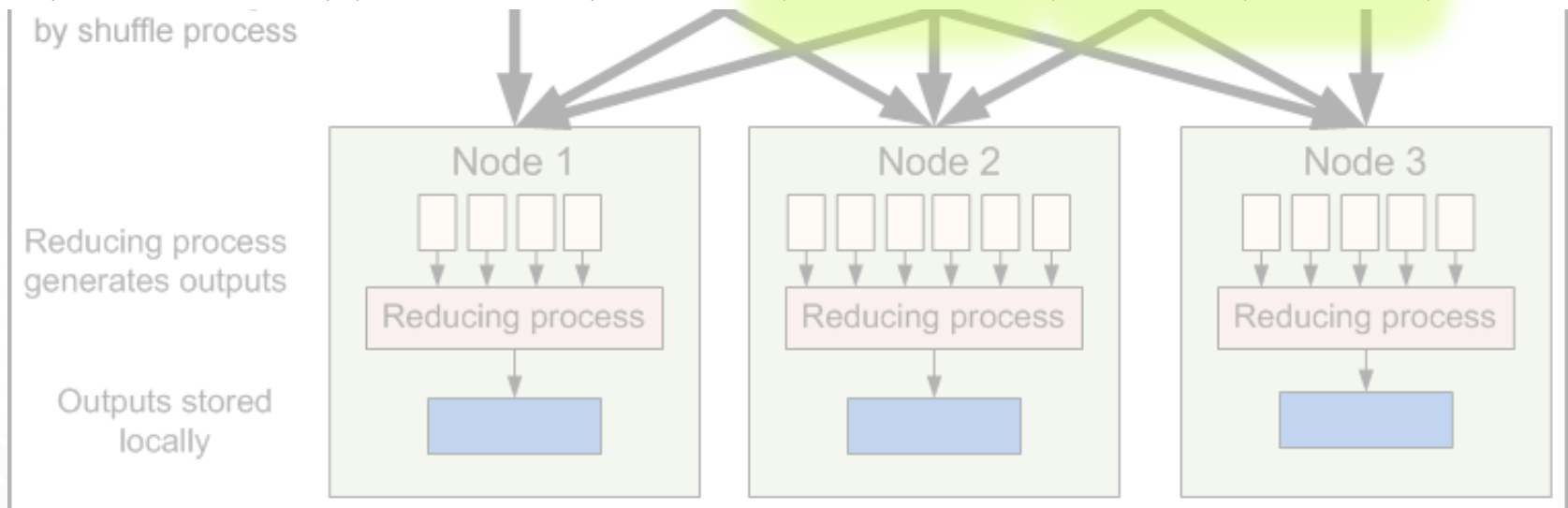
Then Optimize Hadoop

- Channel.write(Message)
- Writable, Text, Mapper\$Context)
- ject, Object, Mapper\$Context)
- MapTask.runNewMapper(JobConf, JobSplit\$TaskSplitIndex, TaskUmbilicalProtocol, Task\$TaskReporter)
- MapTask.run(JobConf, TaskUmbilicalProtocol)
- Child\$4.run()

Combine and Spill Performance



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
ReduceTask.run(JobConf job, TaskUmbilicalProtocol	101706.40	wait (82%) io	194	task[2]@domu-12-31-3	Default Applicati
ReduceTask.run(JobConf job, TaskUmbilicalProtocol	132191.03	sync wait (50%) io	200	task[2]@domu-12-31-3	Default Applicati

PurePath Hotspots

Time [ms] vs Exec [ms]

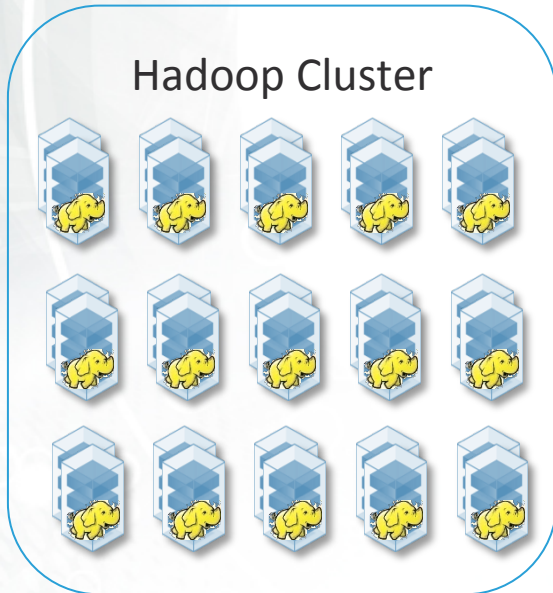
PurePaths Contributors Errors

PurePath Tree (showing only relevant nodes) Show all nodes

Method	Argument	Exec Total [ms]	Elapsed Time [ms]	Breakdown	Class
runNewReducer(JobConf job, TaskUmbilicalProtocol umbilical, TaskUmbilicalProtocol umbilical)		39158.79	93001.84	cpu (38%) io (62%)	ReduceTask
create(Path f, FsPermission permission, boolean overwrite, int replication)		412.80	93008.26	io (95%)	NativeS3FileSystem
getFileSize(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 getObjectMetadataRequest)	tmp.dynatrace:result/JSlow1/part-r-00000	232.60	93008.52	io (96%)	AmazonS3Client
executeMethod(HostConfiguration hostconfig, HttpMethod httpMethod, String url)	https://tmp.dynatrace.s3.amazonaws.com	185.47	93052.47	io (96%)	HttpClient
exception	Not Found	-	93240.69		AmazonS3Exception
getObjectMetadata(String bucketName, String key)	tmp.dynatrace:result/JSlow1/part-r-00000	67.89	93241.19	io (95%)	AmazonS3Client
getObjectMetadata(GetObjectMetadataRequest getObjectMetadataRequest)	tmp.dynatrace:result/JSlow1/part-r-00000	67.86	93241.20	io (95%)	AmazonS3Client
executeMethod(HostConfiguration hostconfig, HttpMethod httpMethod, String url)		67.10	93241.74	io (96%)	HttpClient
exception		-	93308.92		AmazonS3Exception

Error is in Reducer when writing to S3

What comes after Hadoop?



Recommended for You

15 for you based on [items](#) you purchased





Cassandra 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 than 100ms
 - Requires fast read/write on discrete data

Key Cassandra Design Tenants

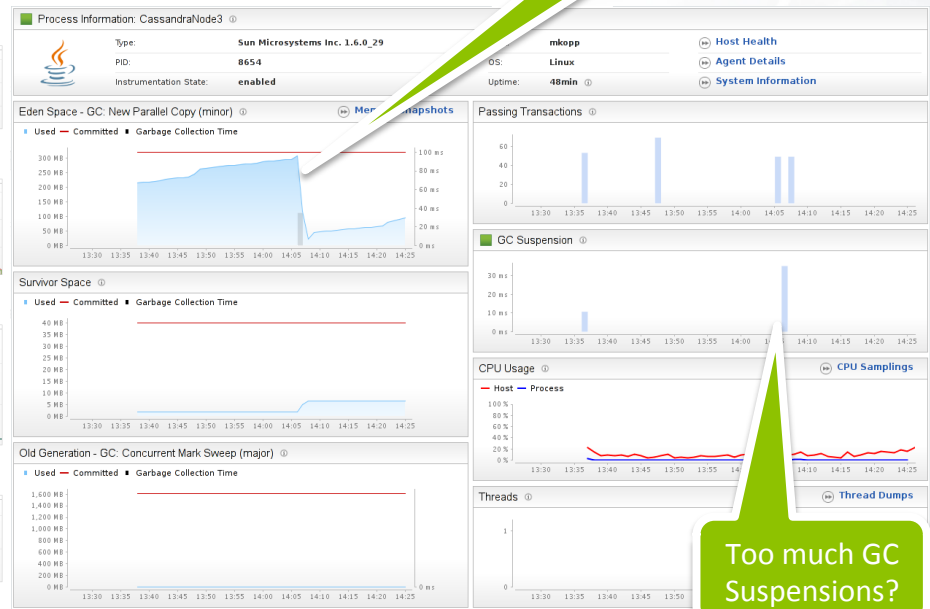
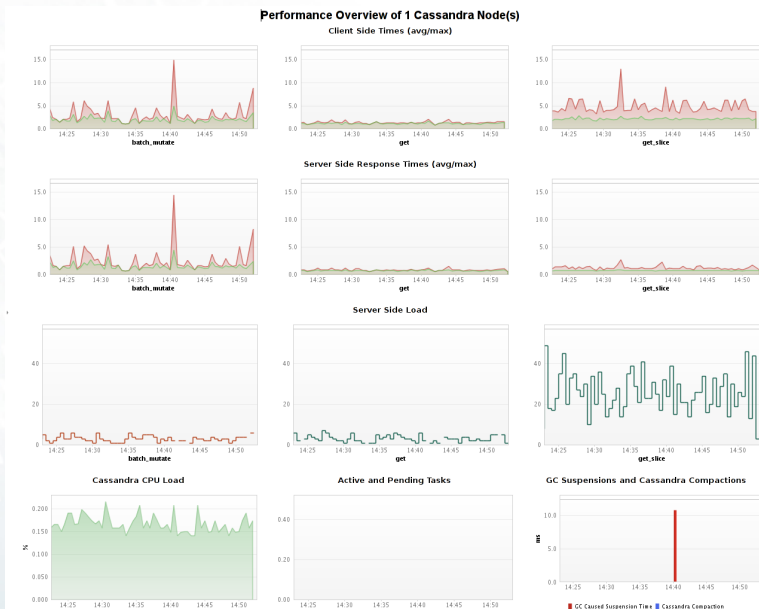
- Swap/paging not possible
- Mostly schema-less
- Writes do not read
 - Read/Write is an anti-pattern
- Optimize around put and get
 - Not for scan and query
- De-Normalize data
 - Attempt to get all data in single read*

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

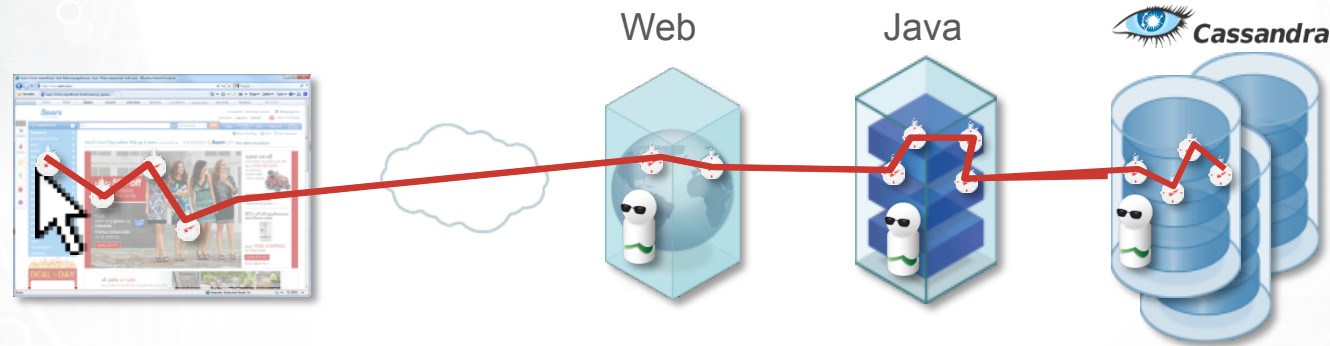


Memory Issues?

Too much GC Suspensions?

APM for Cassandra Applications

NoSQL APM is not so different after all...

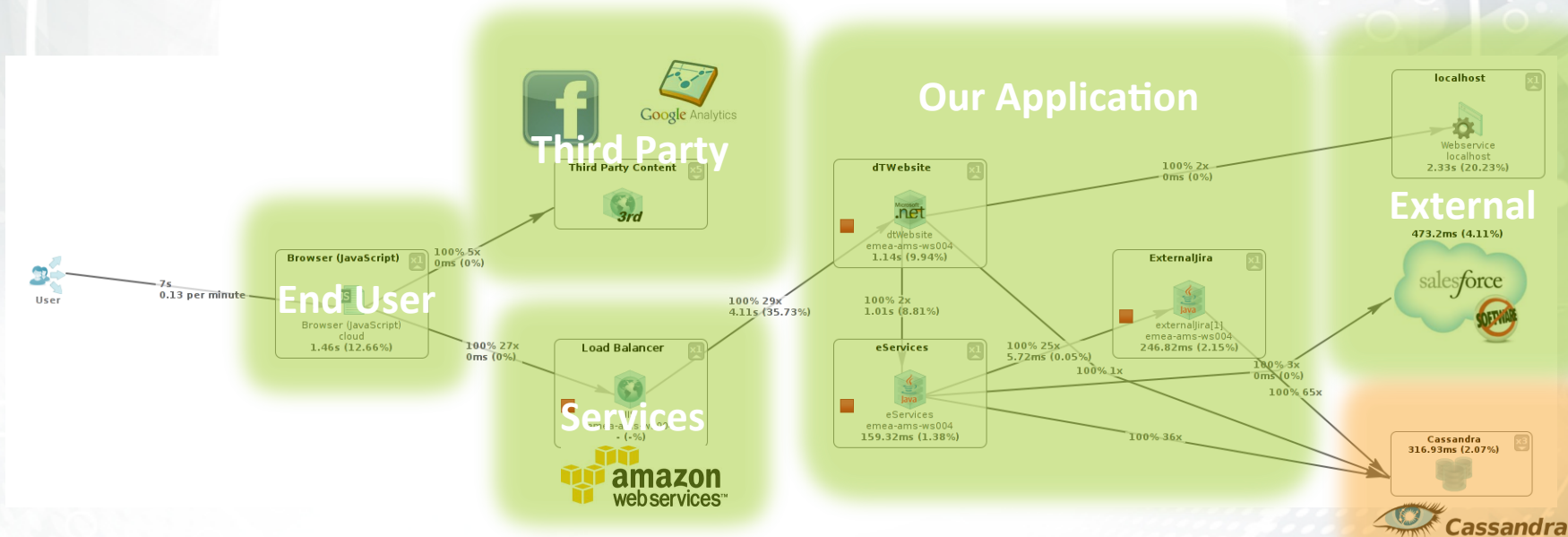


Key APM Problems Identified

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

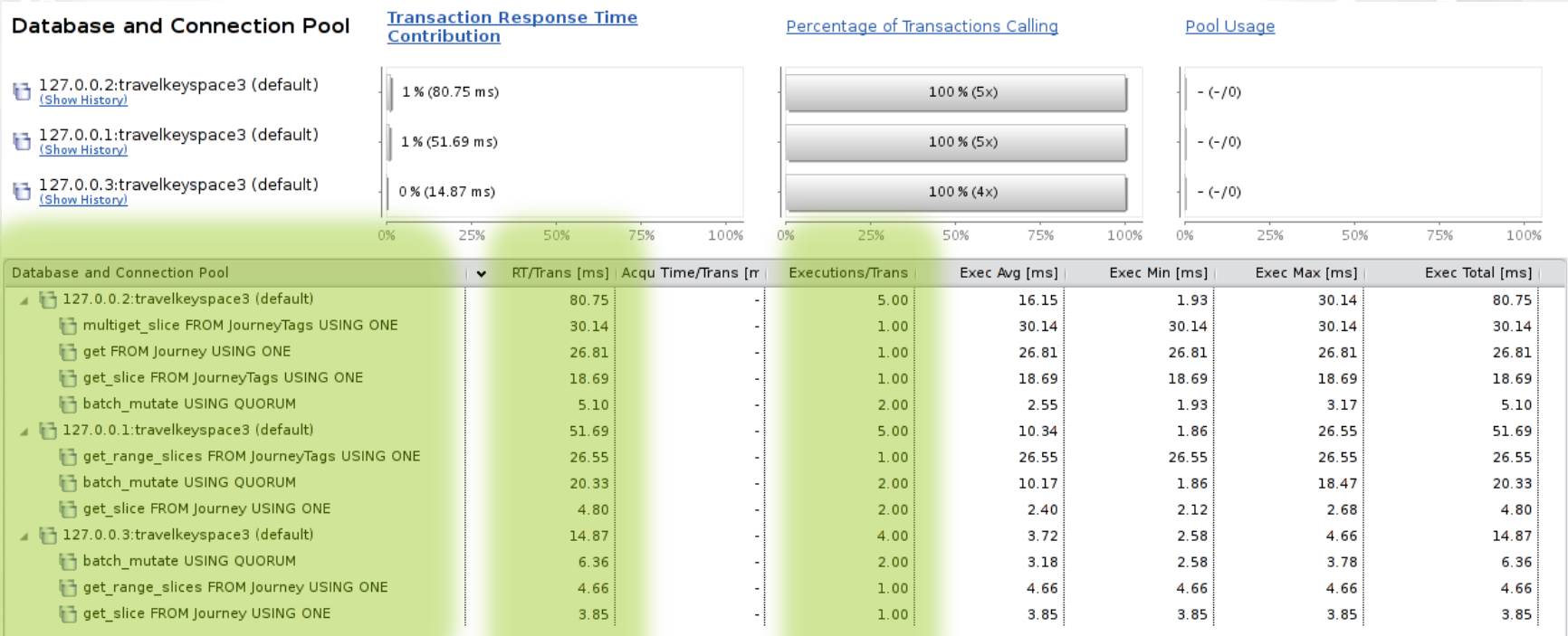
- 1) Data Access Distribution
- 2) End-to-End Monitoring
- 3) Storage (I/O, GC) Bottlenecks
- 4) Consistency Level

Real End-to-End Application Performance



End User Response Time Contribution

Understanding Cassandra's Contribution



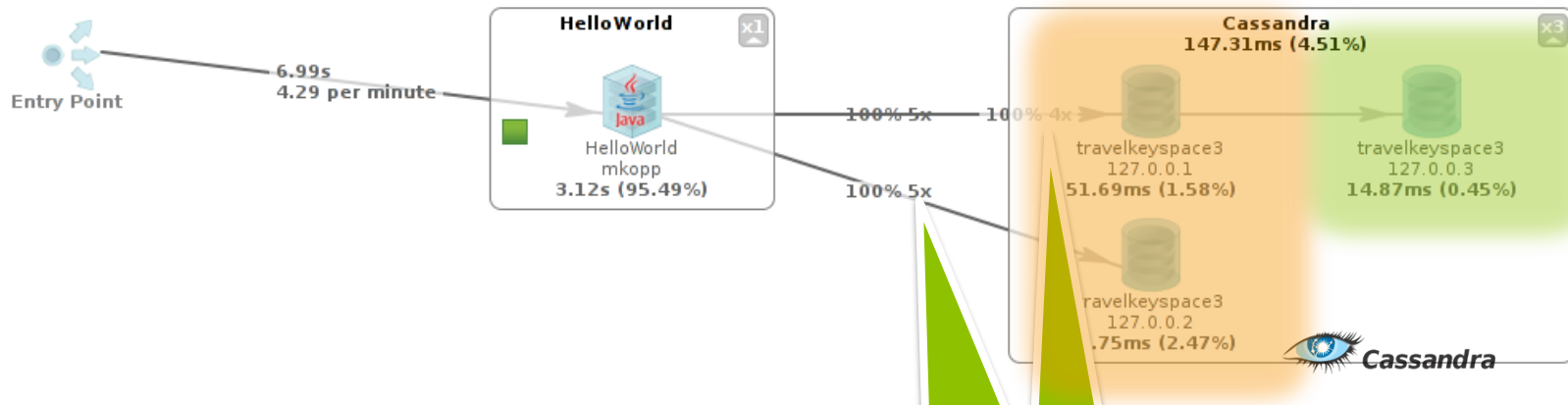
Which statements did the Transaction Execute?
 Which node Contribution of
 Which Consistency Level was used?

Too many calls?

Data Access patterns

Understand Response Time Contribution

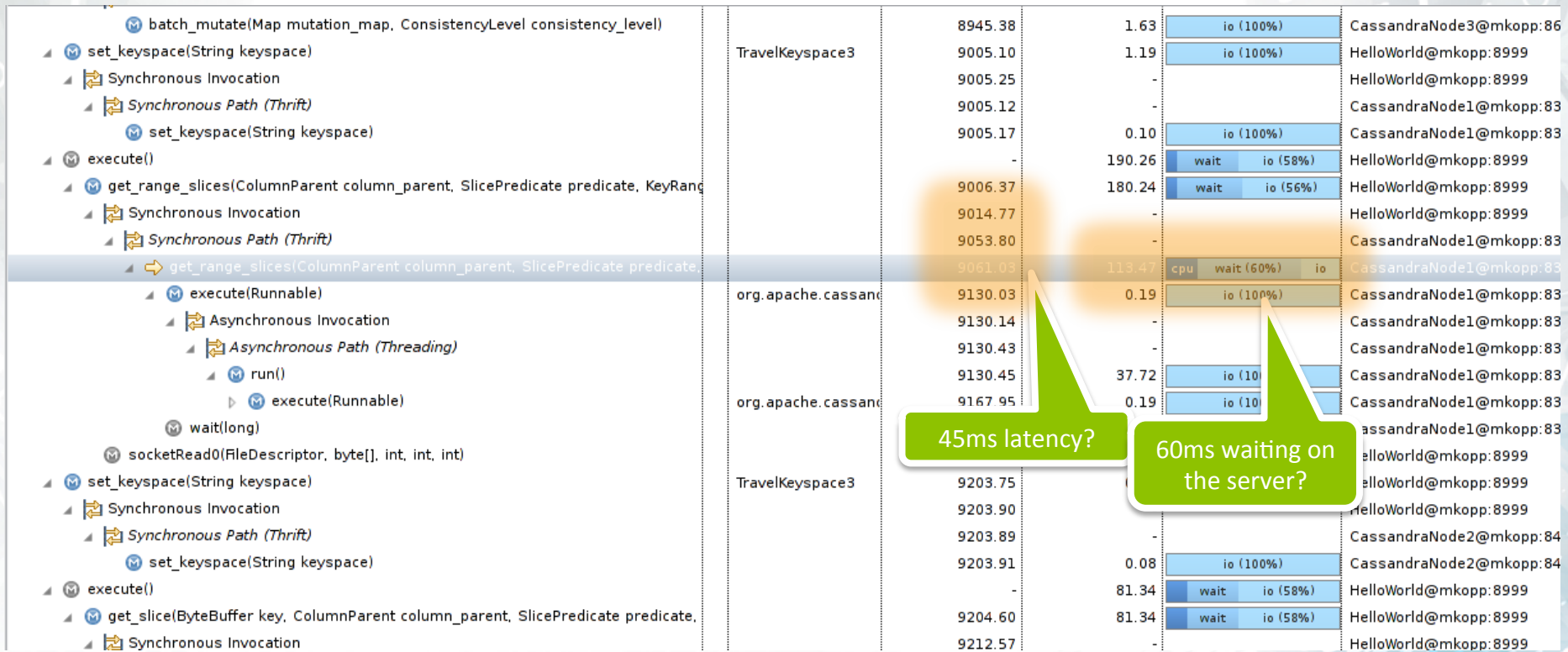
Visualization Mode : Transaction Response Time ▼ ?
 Total Transactions : 1 (4.29 per minute)
 Business Transaction Filter: [no Filter](#)
 Failed Transactions : 0 (0 %)
 Inter Tier Time Per Transaction: 0ms (0%) [show](#)



5 Calls
~50-80 ms Contribution?

Access and Data Distribution

Why and how was a statement executed?



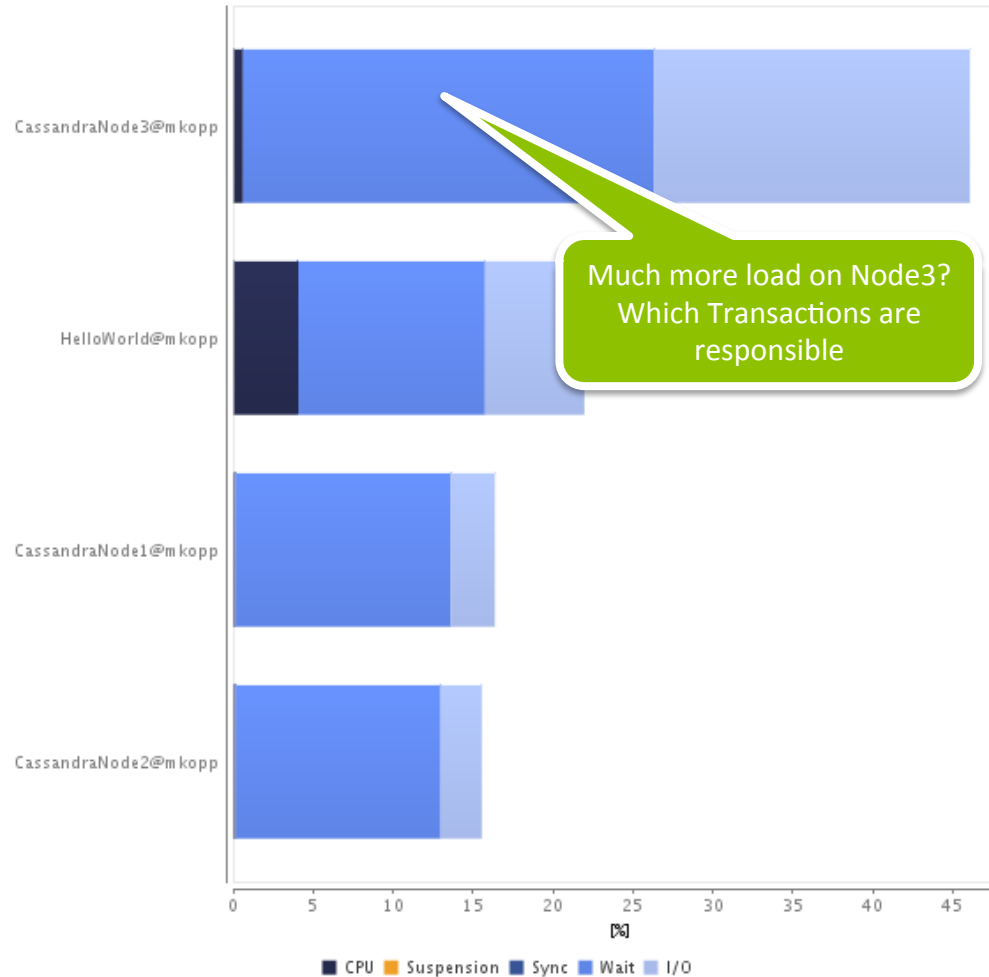
45ms latency?

60ms waiting on the server?

Any Hotspots on the Cassandra Nodes?

Hotspots by Tier

show by Agent ▾

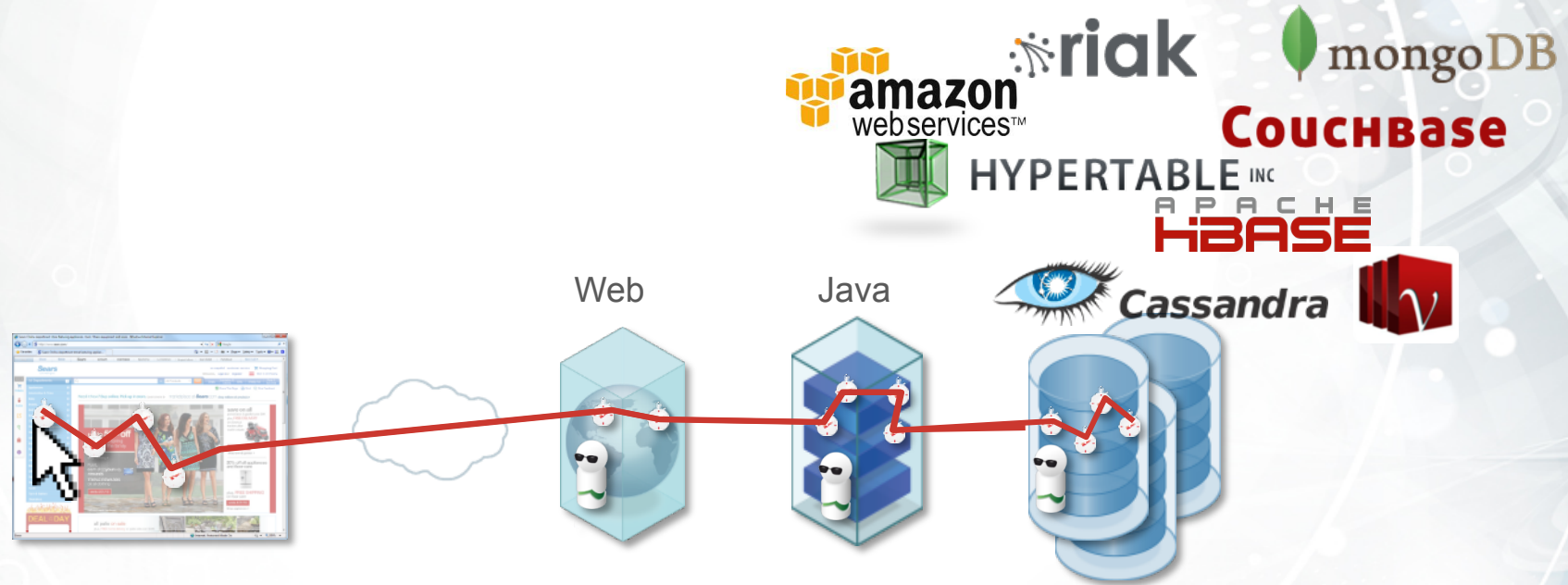




COMPUWARE APM
DYNATRACE® | GOMEZ®

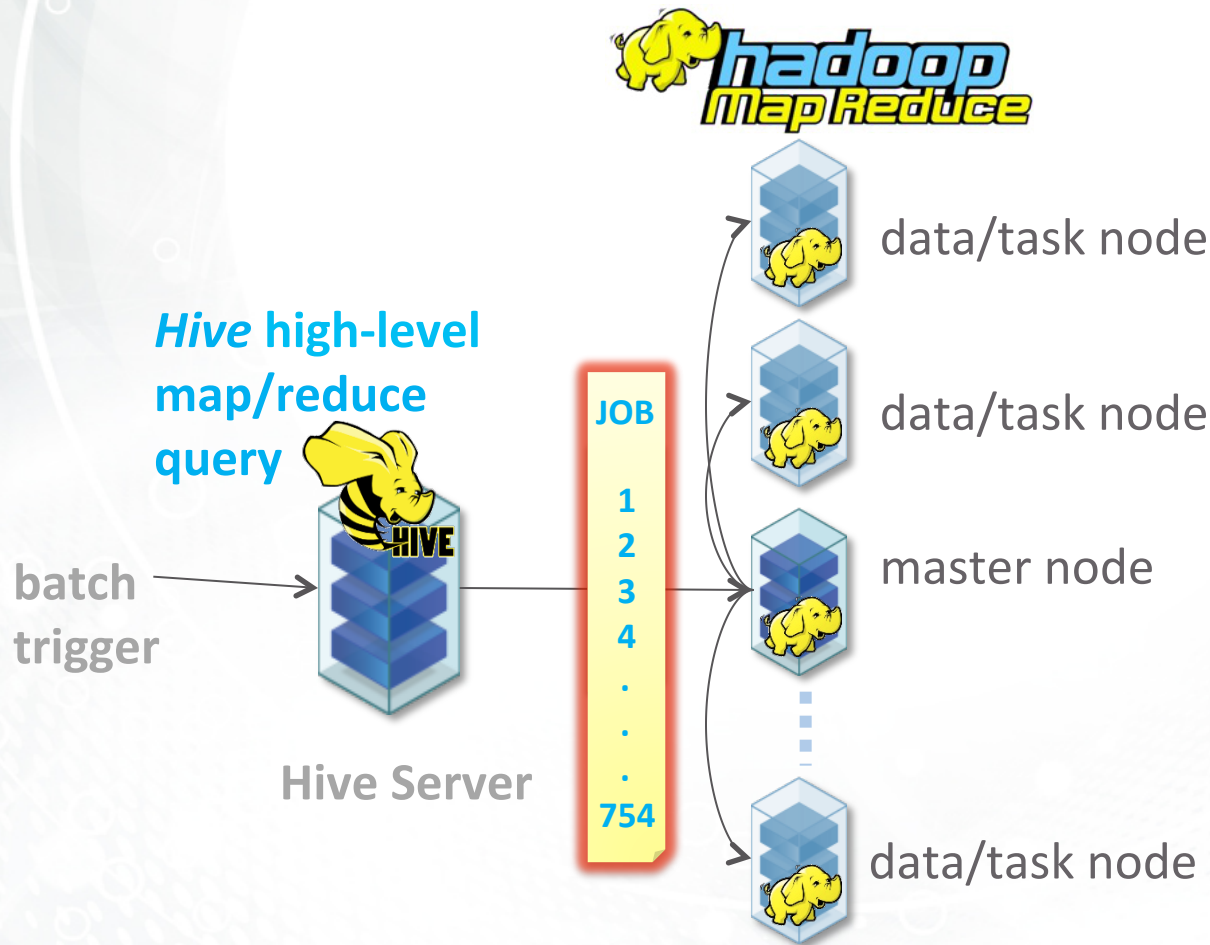
Conclusion

Extend Performance Focus on Application



**A Fast Database doesn't make
a fast Application**

Intelligent MapReduce APM



Simple Optimizations with big impact

Q&A





THE TECHNOLOGY
PERFORMANCE COMPANY

Michael Kopp, Technology Strategist

michael.kopp@compuware.com

@mikopp

blog.dynatrace.com

book.dynatrace.com

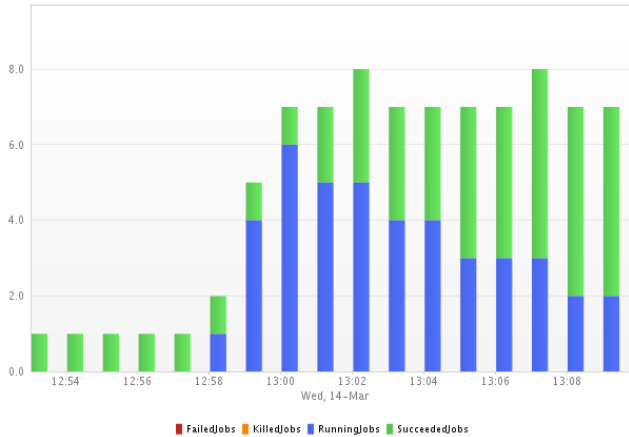
Monitor MapReduce and Hadoop



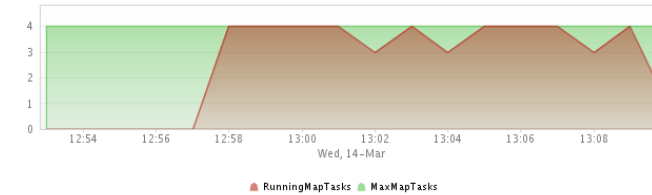
2 Task Trackers are online
1 Jobs are running



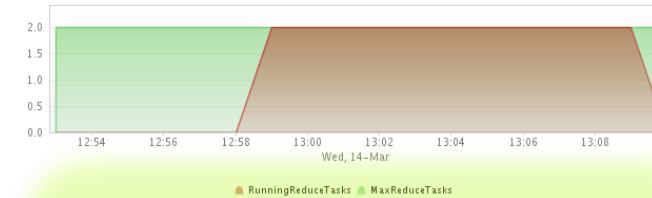
MapReduce Jobs



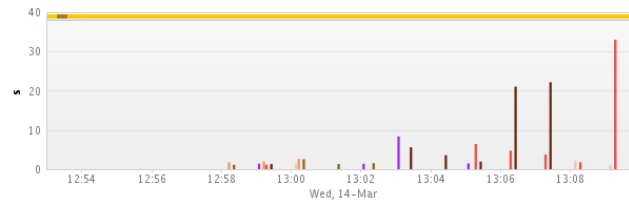
Map Tasks



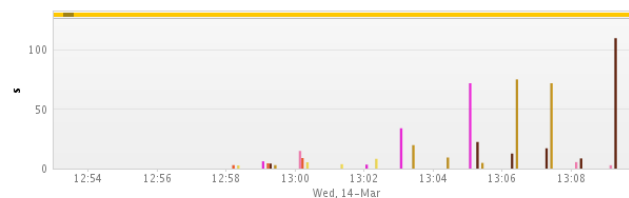
Reduce Tasks



Job CPU Times



Job Times



Business Transaction		from yesterday 12:52 to yesterday 13:09		
Name	Count	Failed %	PurePath	Response Time [s]
Map/Reduce Job (split by Task)	219	0.91 %		2275771.71 ms
job_201203141133_0005	38	2.63 %		707581.83 ms
job_201203141133_0006	35	0 %		657596.63 ms
job_201203141133_0004	37	0 %		307290.47 ms
job_201203141133_0002	38	2.63 %		285910.03 ms
job_201203141133_0003	38	0 %		174023.14 ms
job_201203141133_0007	33	0 %		143369.60 ms

Configuration problems were detected on Server 'domu-12-31-39-0e-95-53'. Click here for details.