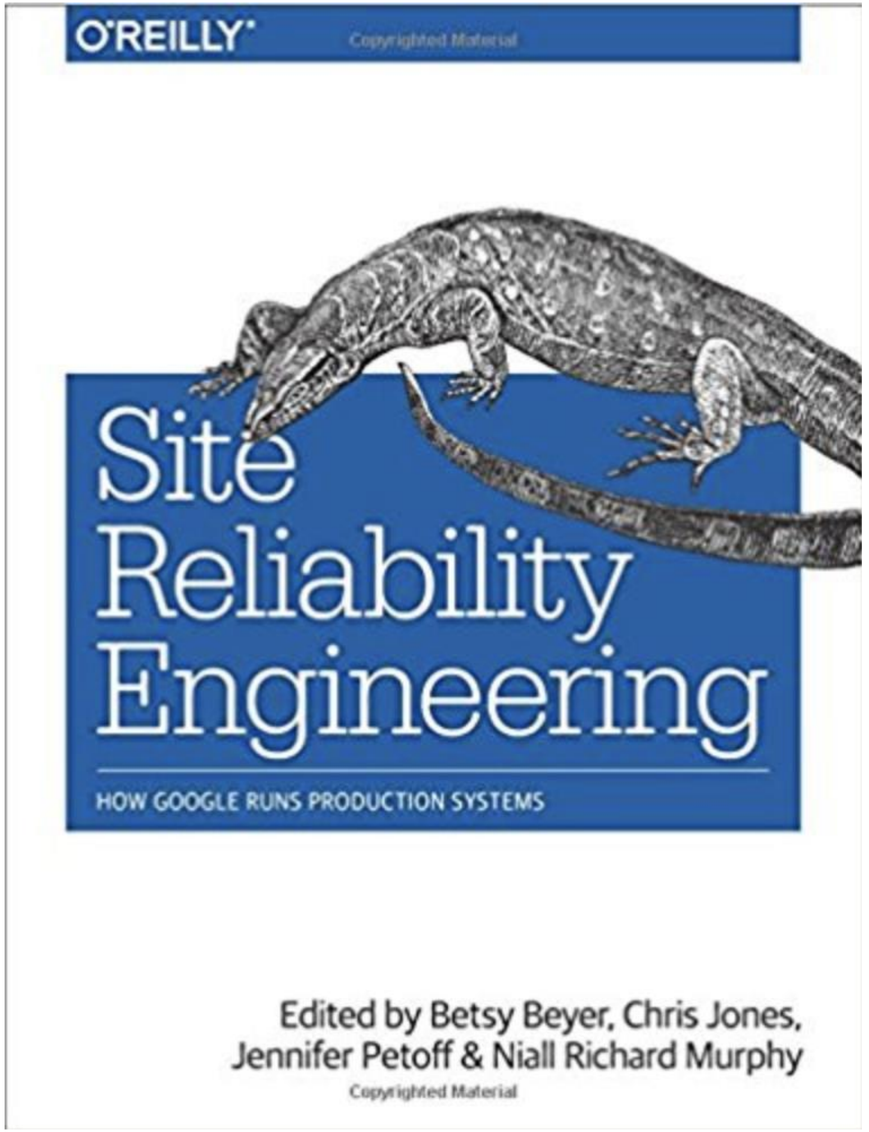
William Montaz & Nicolas Fraison
R&D - Lake

Hadoop cluster perf

Operating one of the largest clusters in Europe



Criteo Hadoop Cluster

3k datanode/nodemanagers

337,179,866 files and directories, 316,183,533 blocks = 653,363,399 total filesystem object(s).

Heap Memory used 265.91 GB of 540.83 GB Heap Memory. Max Heap Memory is 540.83 GB.

Non Heap Memory used 60.79 MB of 256 MB Commited Non Heap Memory. Max Non Heap Memory is 256 MB.

Configured Capacity:	223.5 PB
DFS Used:	158.88 PB (71.09%)
Non DFS Used:	12.82 TB
DFS Remaining:	64.59 PB (28.9%)
Block Pool Used:	107.83 PB (48.25%)
DataNodes usages% (Min/Median/Max/stdDev):	7.73% / 75.96% / 83.46% / 9.60%
Live Nodes	2805 (Decommissioned: 11, In Maintenance: 0)

HDFS (storage)

Namenode Heap 500Go

220 PB RAW Storage / 150 PB RAW Data used

Around 714 millions of blocks splitted on 3 namespaces

Around 25 millions of blocks created and removed every day

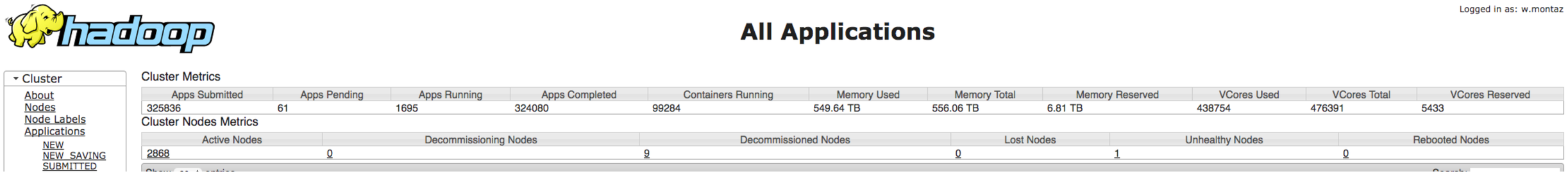
Criteo Hadoop Cluster

3k datanode/nodemanagers

YARN (scheduling)

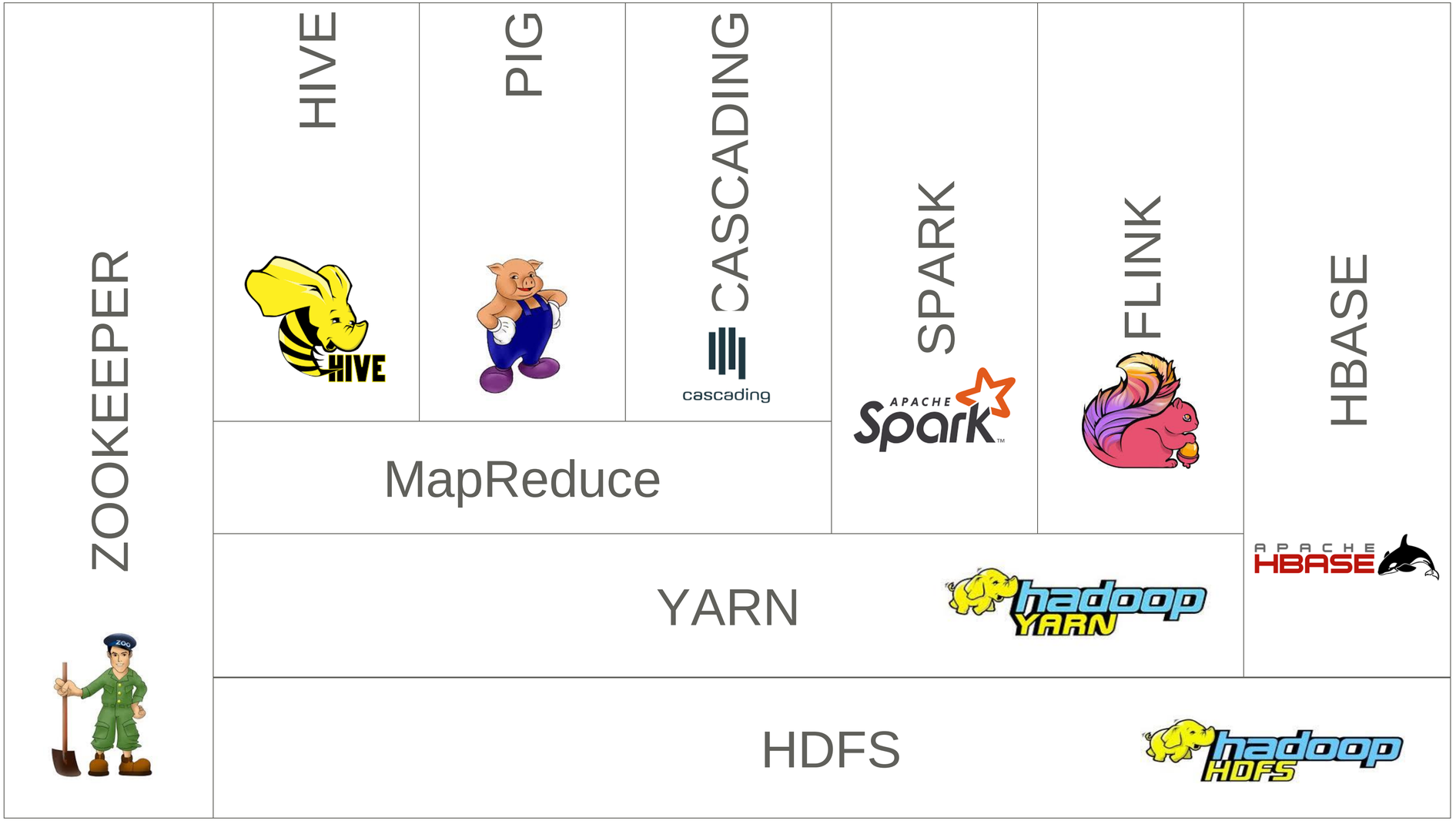
550 TB of Memory and 72k cores

300k jobs/day executing 150 millions of containers/day



Platform Scaling

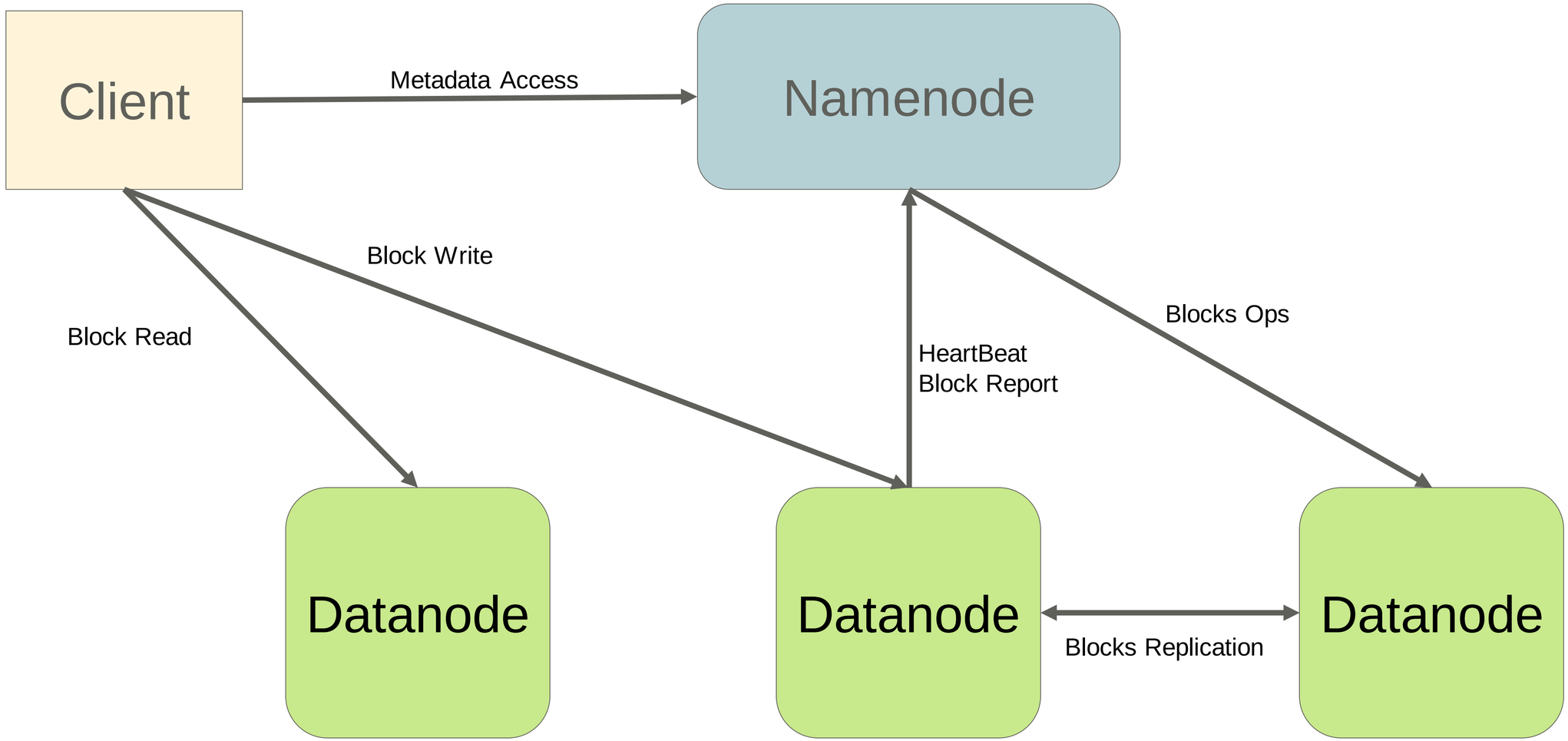
HADOOP Big Picture



criteo

Namenode Perf

HDFS Architecture



Namenode – In Memory Structure – G1

Files/Directories and Blocks metadata in memory (1 Millions blocks around 700Mo of heap)

Datanodes metadata

HDFS Token

Metadata objects are long lived objects

Namenode – In Memory Structure – G1

Mixed are launched with young

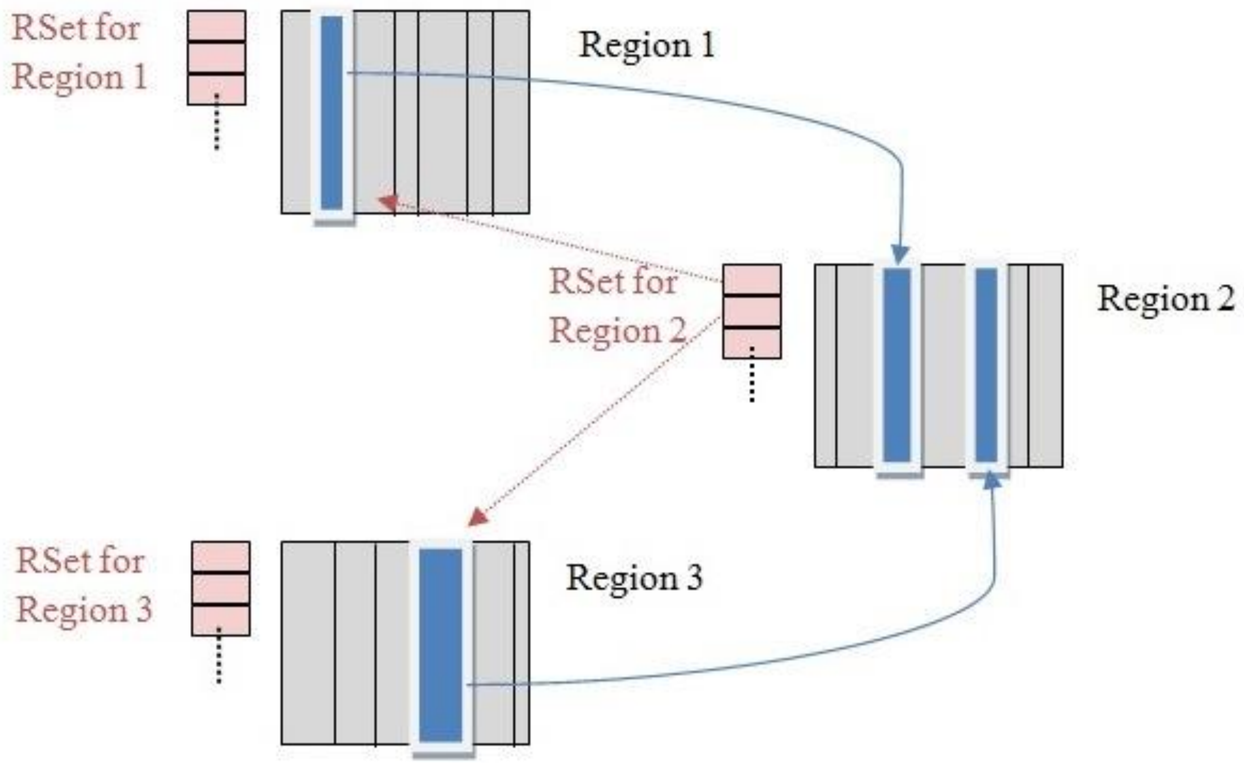
Be carefull tuning G1NewSizePercent

G1 concentrate collection on more reclaimable regions

Namenode – In Memory Structure

Update RS and Scan RS impact STW phase duration

```
    [Update RS (ms): Min: 3382.1, Avg: 3382.4, Max: 3382.8, Diff: 0.7, Sum: 74413.2]
        [Processed Buffers: Min: 5013, Avg: 5230.0, Max: 5604, Diff: 591, Sum: 115059]
     [Scan RS (ms): Min: 11150.6, Avg: 11150.9, Max: 11151.0, Diff: 0.4, Sum: 200715.5]
```



G1 is cool, still suffers from design choices

-> triggers mixed with young

-> Not robust to application memory profile changes

-> still needs tuning

Always write GC logs

Long-lived and short-lived objects, not in-between

ByteBuffers uses off heap until GC

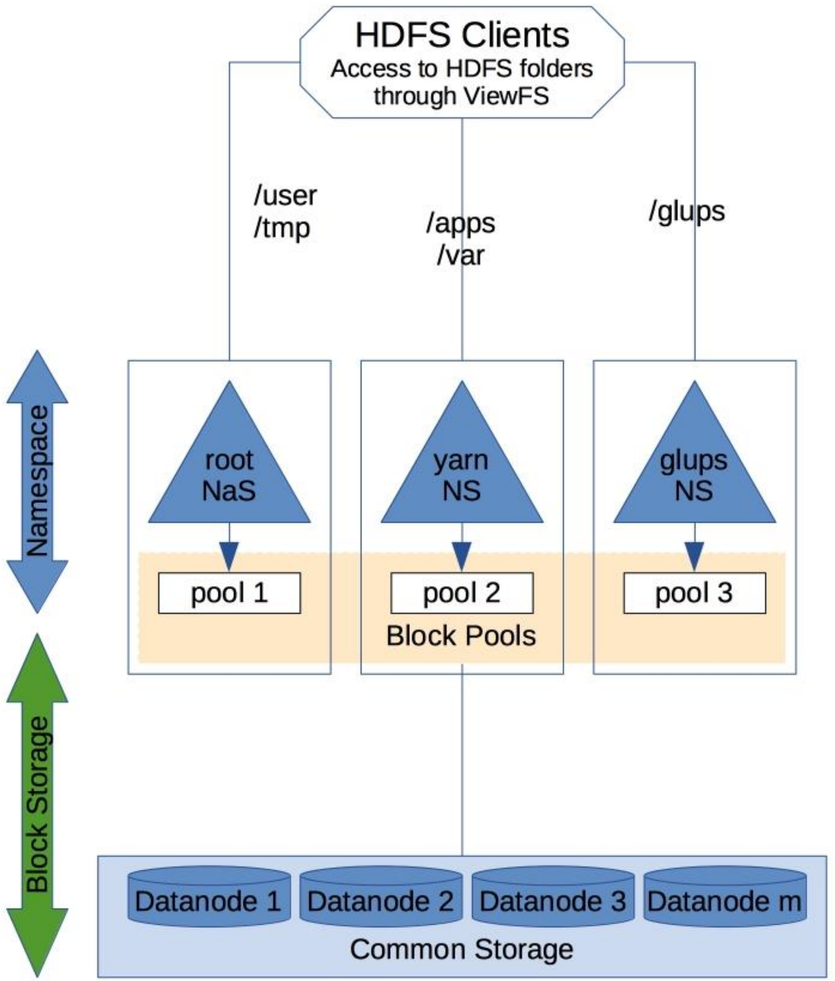
criteo

Namenode - Federation

Use multiple independant Namespaces to scale horizontally

Datanodes are used as common storage for blocks by all Namenodes

ViewFS is used by user to map a folder to a specific namespace



Namenode – Federation

Bad scalability pattern

- Configuration is hosted on clients
- Split to decrease block pressure can be incompatible with split to decrease IOs
- Does not take in account hardware capabilities of namenode per namespace
- Does not take in account load changes
- All datanodes report to all namenode

criteo

Namenode – Federation

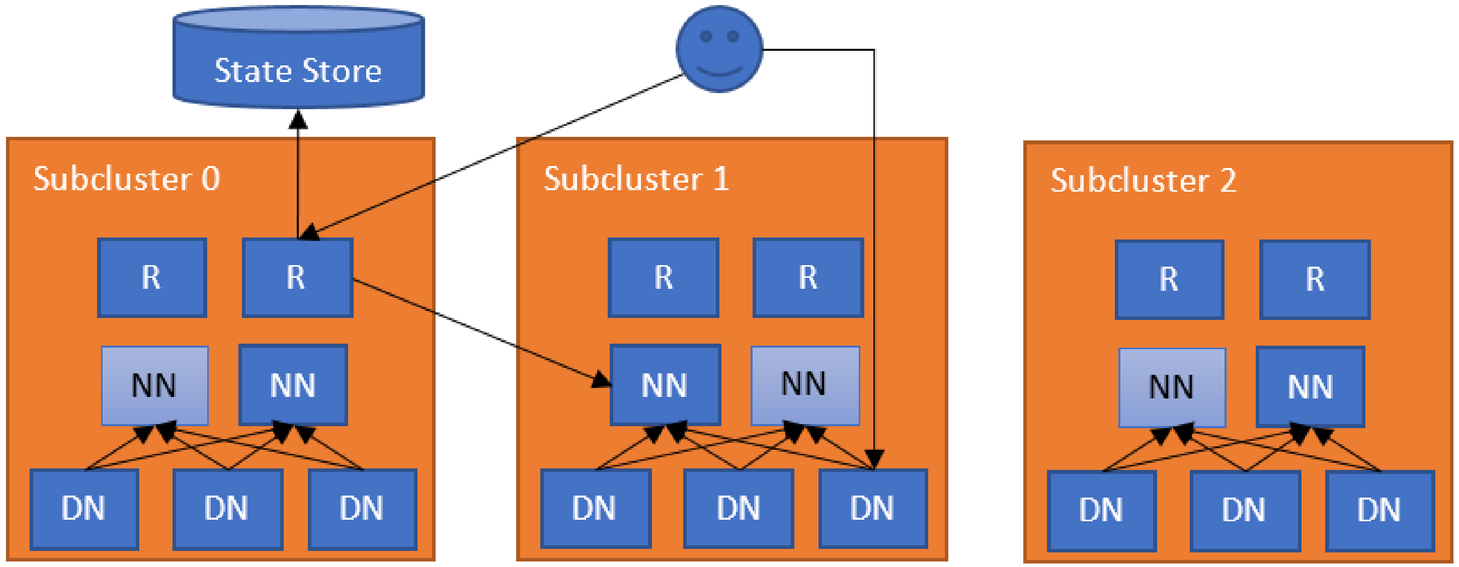
Next Step (HDFS Router-based Federation)

Client only access one endpoint

Router component manage access to appropriate namespace

Router will manage rebalancing of data between namespace

Done on not shared datanode



Carefull with in memory, especially if it is non transient data

Split your app in simple components

Design your app to be multi instance at first

criteo

Namenode – Patch Issue

Issue

Slownesses of all HDFS requests after bumping namenode from cdh5.5 to cdh5.11

How to investigate

GC logs
ThreadDump / Zing vision
Check diff between both release (git log ;))

Results

Memory behaviour changed: a long mixed GC every hour and increase of allocation rate
Long running time of GetContentSummary
Large Hash table allocated

Solution

Revert HDFS-10797 (Disk usage summary of snapshots causes renamed blocks to get counted twice)

Measure don't guess

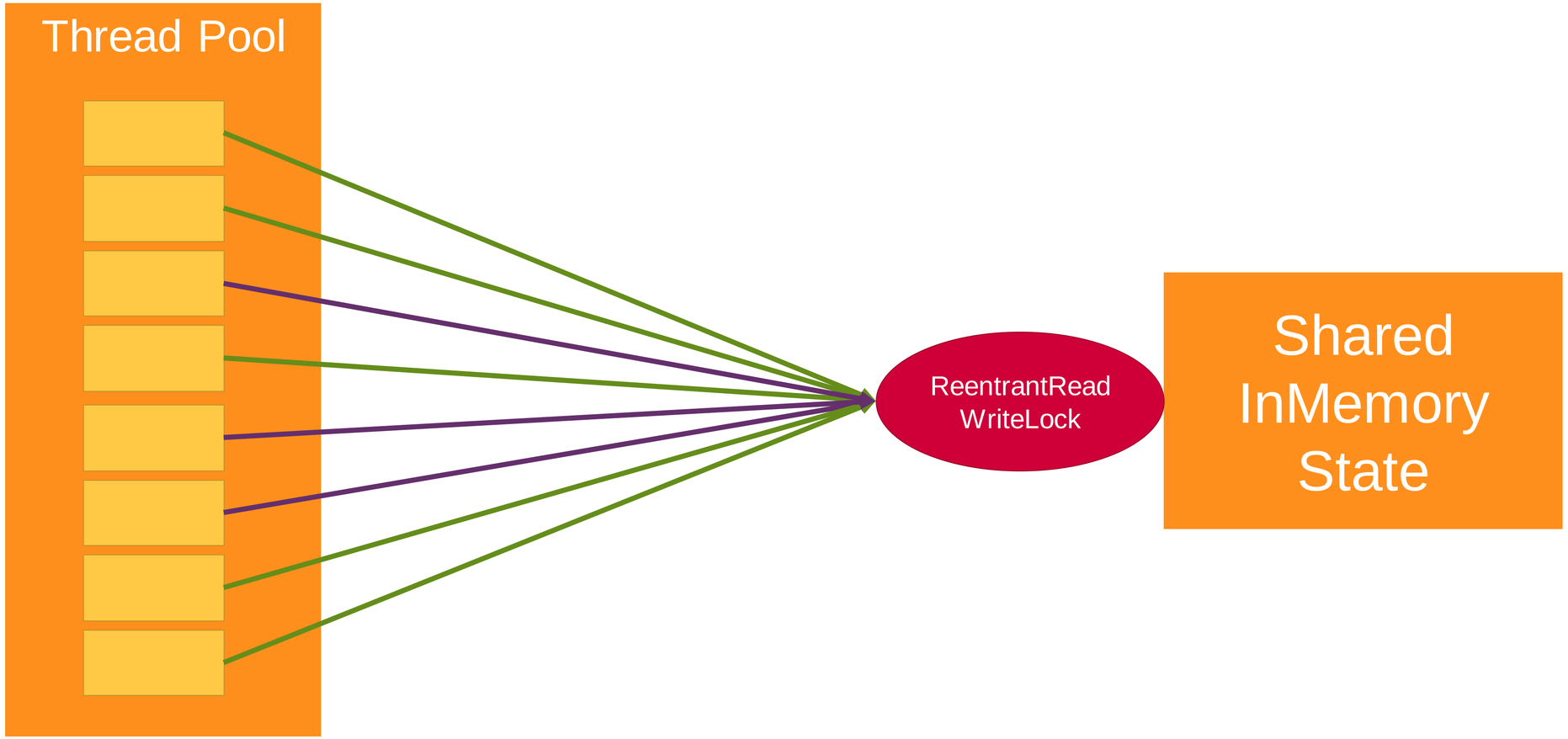
Always GC logs

Always Thread Dump

Have release notes version control based

criteo

Namenode – ReentrantReadWriteLock



criteo

Namenode – ReentrantReadWriteLock

« A nonfair lock that is continuously contended may indefinitely postpone one or more reader or writer threads »

ReentrantReadWriteLock

Namenode – ReentrantReadWriteLock

« Because checks in acquire are invoked before enqueuing, a newly acquiring thread may barge ahead of others that are blocked and queued »

AbstractQueuedSynchronizer

Namenode – ReentrantReadWriteLock

```
/**
 * Fair version of Sync
 */
static final class FairSync extends Sync {
    private static final long serialVersionUID = -2274990926593161451L;
    final boolean writerShouldBlock() {
        return hasQueuedPredecessors();
    }
    final boolean readerShouldBlock() {
        return hasQueuedPredecessors();
    }
}
```

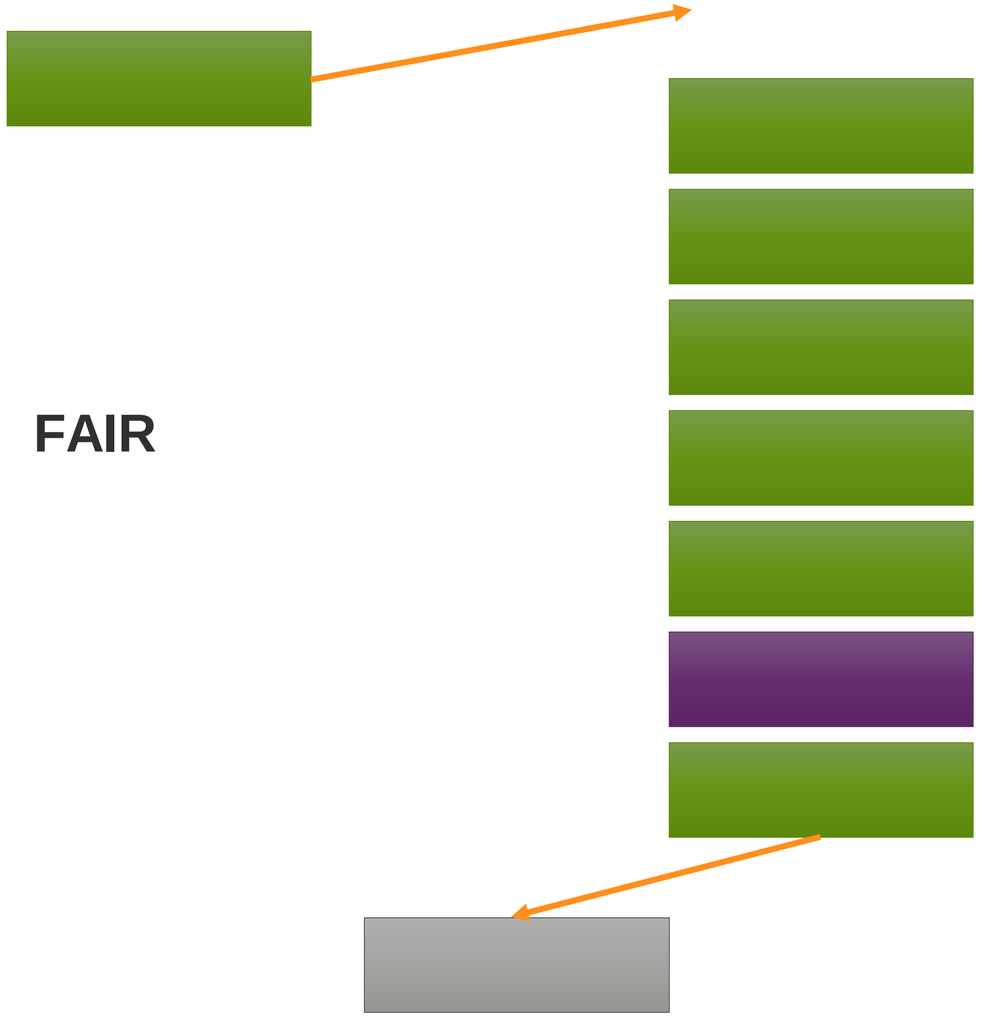

criteo

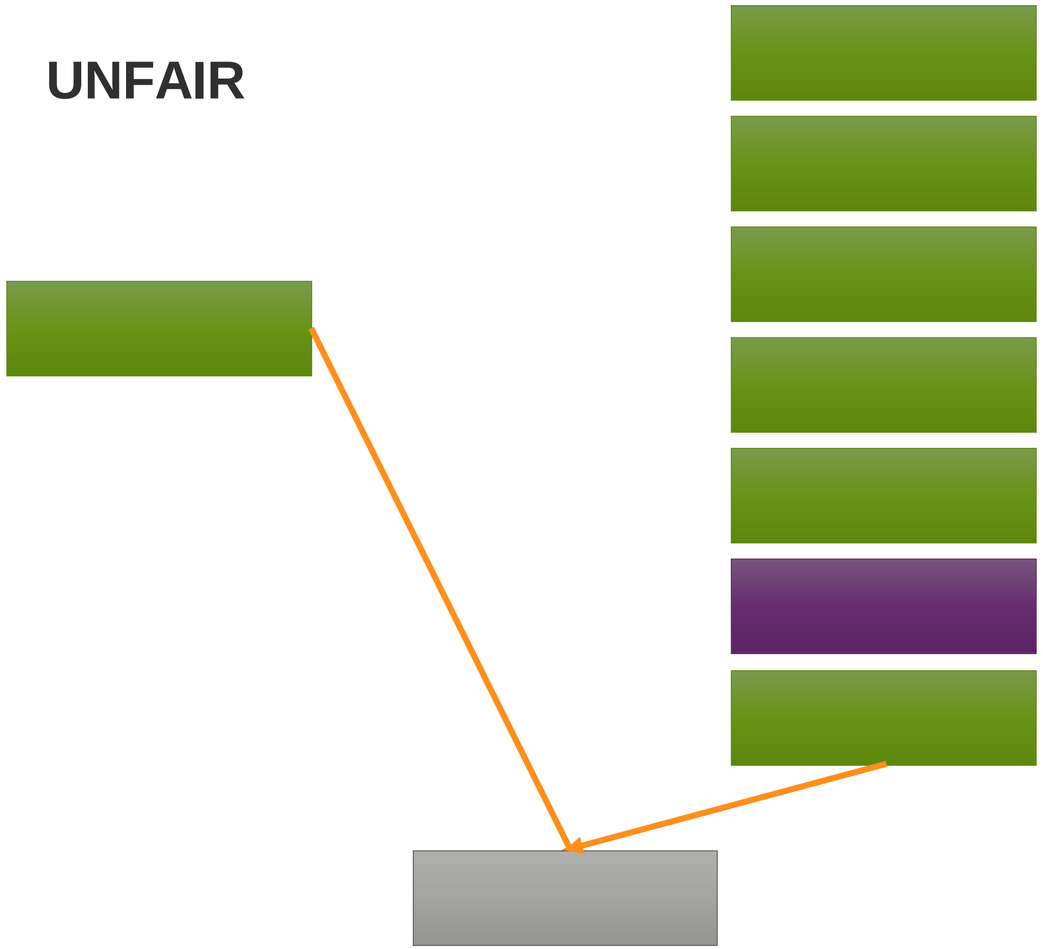
Namenode – ReentrantReadWriteLock

```
/**
 * Nonfair version of Sync
 */
static final class NonfairSync extends Sync {
    private static final long serialVersionUID = -8159625535654395037L;
    final boolean writerShouldBlock() {
        return false; // writers can always barge
    }
    final boolean readerShouldBlock() {
        /* As a heuristic to avoid indefinite writer starvation,
         * block if the thread that momentarily appears to be head
         * of queue, if one exists, is a waiting writer.  This is
         * only a probabilistic effect since a new reader will not
         * block if there is a waiting writer behind other enabled
         * readers that have not yet drained from the queue.
         */
        return apparentlyFirstQueuedIsExclusive();
    }
}
```


criteo

Namenode – ReentrantReadWriteLock





criteo

Namenode – ReentrantReadWriteLock

Fair mode

- Prevent many possible read parallelisation
- Introduces more thread parking
- Less variance

Try unfair first !

criteo

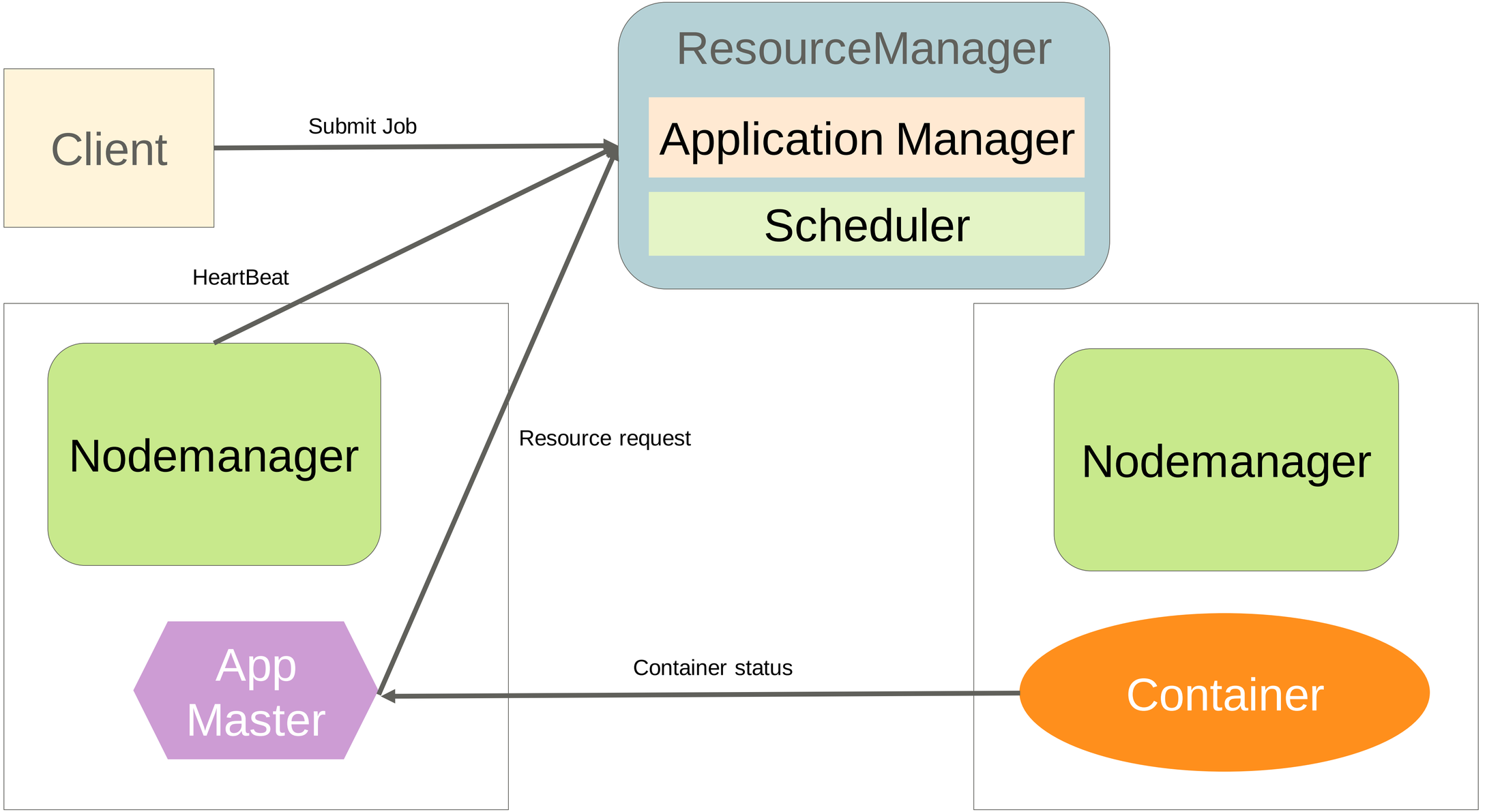
Truth is in the source code, verify other sources against it

criteo

ResourceManager Perf

criteo

YARN Architecture



Resourcemanager

Issue

Slownesses and OOM of the Resourcemanager

How to investigate

GC logs

Async-profiler (https://github.com/jvm-profiling-tools/async-profiler)

Async-profile:

Low overhead profiling (CPU, Heap allocation)

Attached on the fly to the JVM

Does not suffer from Safepoint bias problem

Resourcemanager

Investigation Results

Lots of time spend on GC with high allocation rate (6G/s)

Heap Allocation (2 abnormal pattern)

Solutions

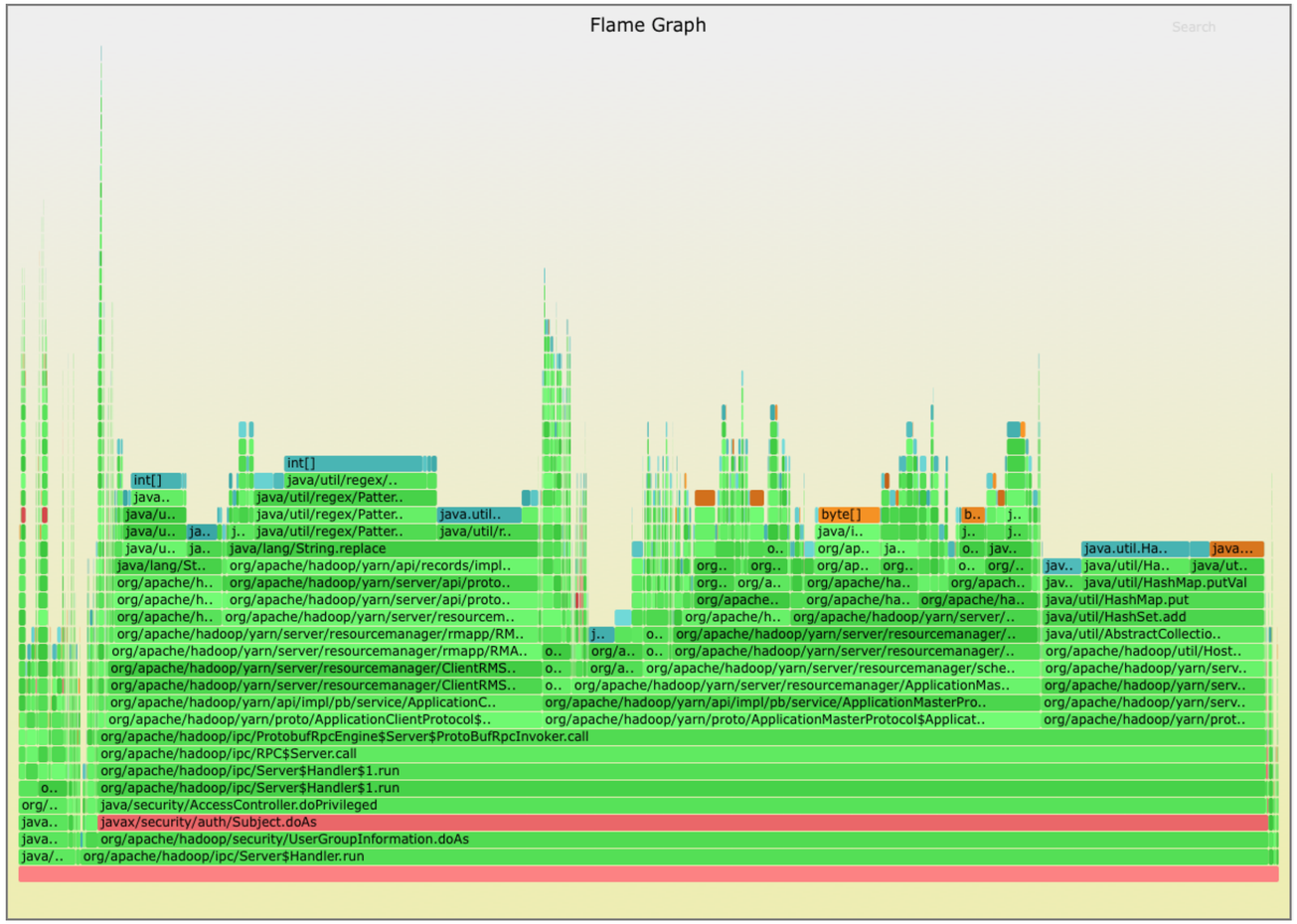
Reduce nodemanager heartbeat frequence

Apply patches

Review usage of getApplications API

Results

Allocation drastically reduced to 500 Mo/s



Monitor Allocation Rate

Remember Hardware limits (bus bandwidth)

Protect dangerous API endpoints

Monitor API usage and help users to get the best of it

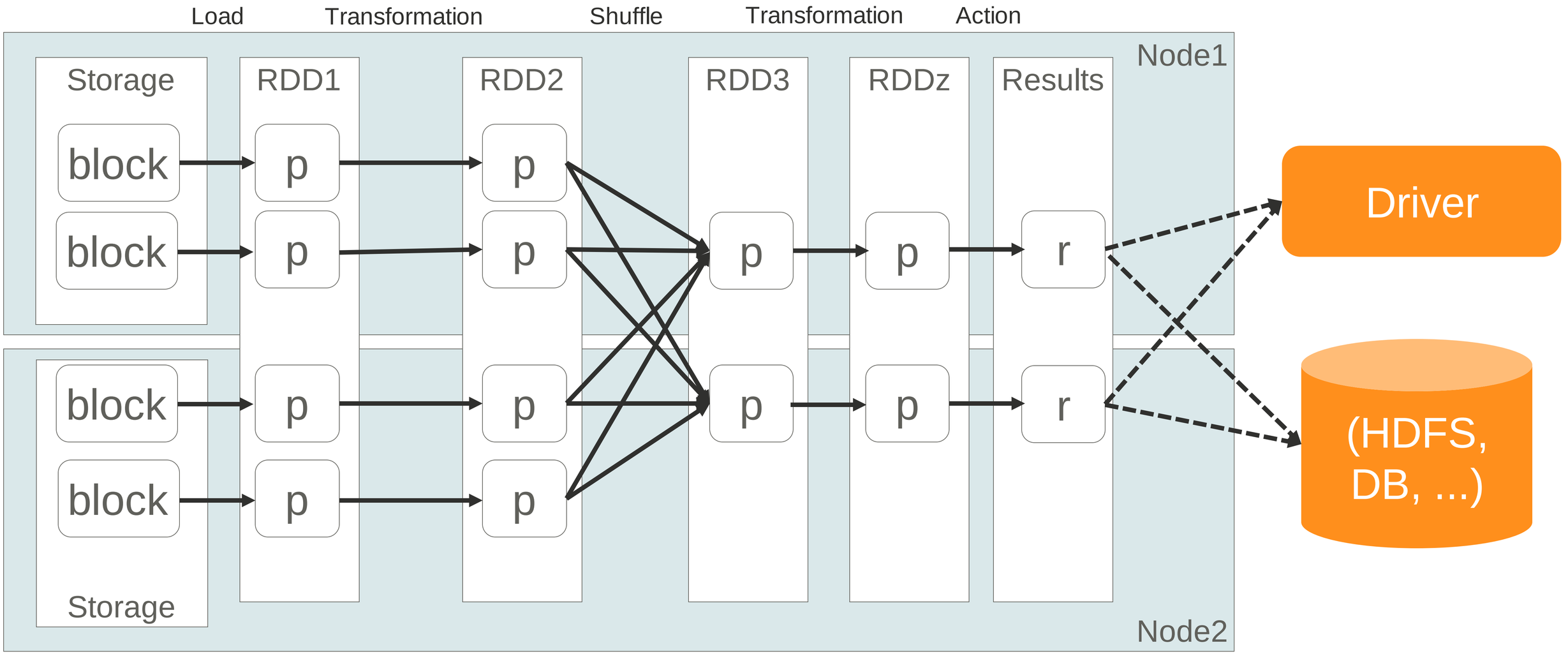
Use AsyncProfiler, but be careful

criteo

Shuffle Step

criteo

Spark – RDDs operation



Spark Shuffle

Issue

Some spark learning jobs take 10* more time than other

Investigation

One executor treat up to 50% of partitions

Uneven distribution of RDDs

Connection timeout issues during shuffle phase

Connection issue leads to uneven distribution of RDDs

Spark shuffle service stuck transfering data and not accepting new connections

```
2018-02-03 04:48:29,021 ERROR
org.apache.spark.network.server.TransportChannelHandler:
Connection to host1 has been quiet for 600000 ms while there
are outstanding requests. Assuming connection is dead; please
adjust spark.network.timeout if this is wrong.
```


criteo

Spark Shuffle

Solution

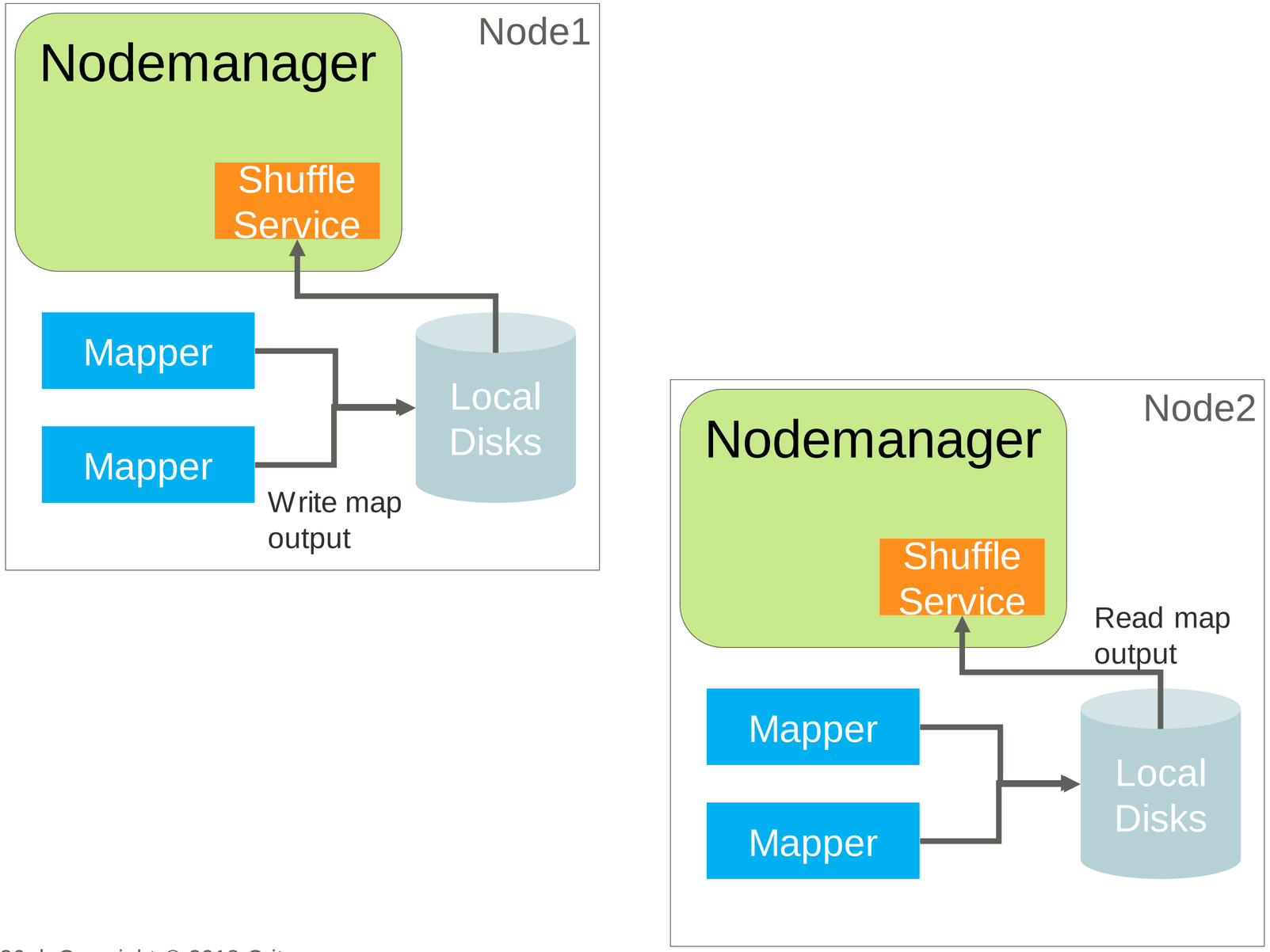
- Increase backlog size shuffle.io.backLog
- Increase number of shuffle threads shuffle.io.serverThreads

ToDo

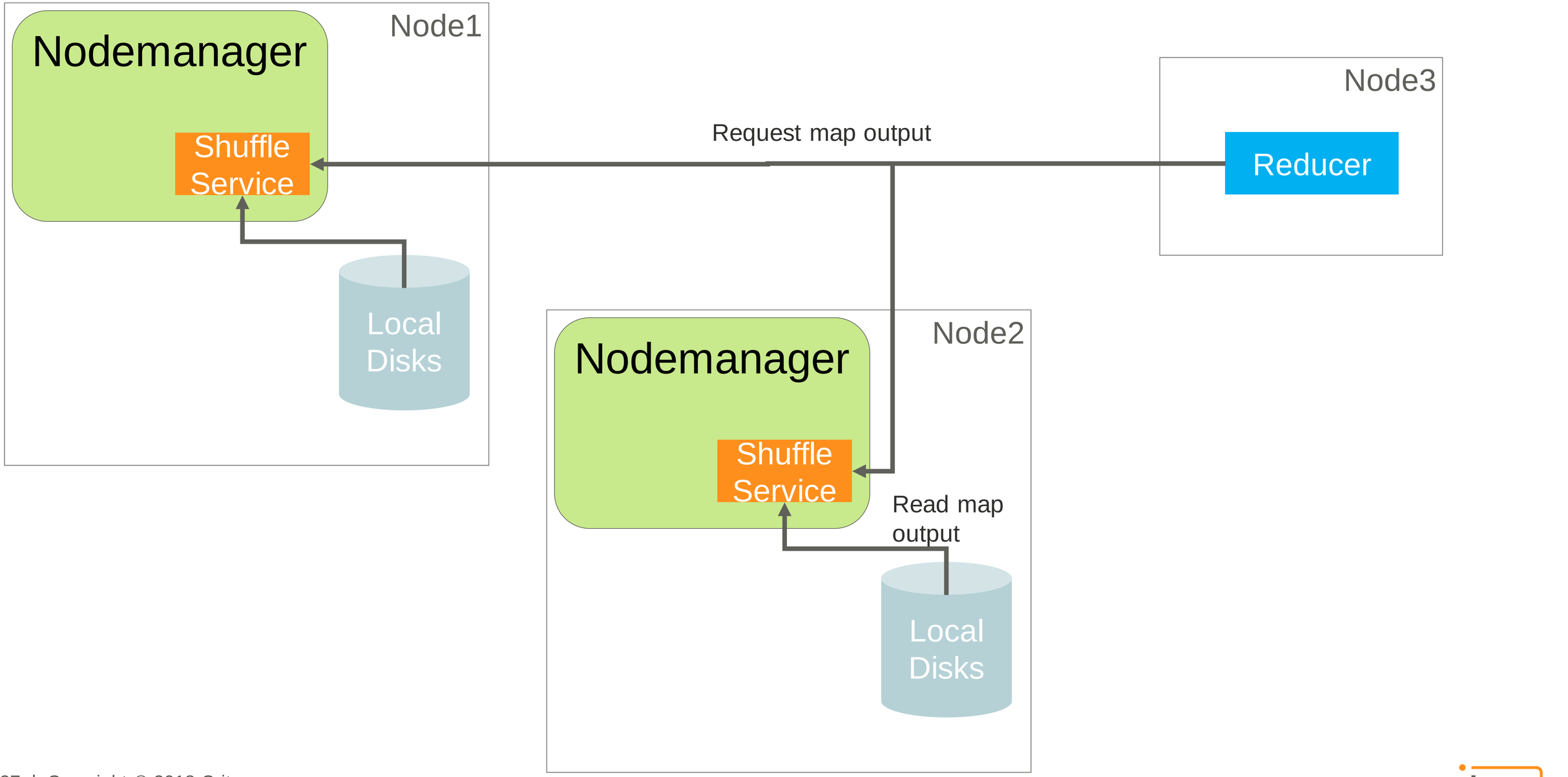
- Split acceptor and worker loop threadpool

criteo

MapReduce Shuffle Service issue



MapReduce Shuffle Service issue



MapReduce Shuffle Service issue

Issue

- Bigger MapReduce jobs are failing due to *Too many fetch failures*

Investigation

- Fetch issue on reduce phase only when trying to get map output
- No issue from shuffle service logs on nodemanager
- TCP backlog queue of shuffle service socket full when issue happen

Solution

- Increase net.core.somaxconn
- Configure shuffle queue socket size (mapreduce.shuffle.listen.queue.size)

criteo

Monitor TCP stack

Leave acceptor thread alone

Deal with connection spikes with TCP backlog

Manage failures client side

criteo

2

Cluster Measurement and Analysis

Slow Nodes

criteo

Slow nodes

Nodes eventually become slow

Over-provisionning of our nodes
OS become slow
Hardware issue

Only one slow nodes can affect all of our big jobs

We cannot execute full platform rolling restarts (for now)

So, We need to pinpoint slow nodes and apply fix

Slow nodes

Hadoop solution: Speculative execution

Execute more containers than needed, get results from the fastest ones

Why we don't like it ?

Not compatible with some of our old learning jobs that can’t have double run of the same tasks at the same time

Waste of resources on the cluster

Even worst with heterogenous hardware !

Framework dependent

Slow nodes

What do we do ?

Daily metrology pipeline analyzing logs for mapreduce/spark exports to Vertica DB

Make a node ranking (0 to 100) per job execution

Calculate mean of the ranks per node

If mean > 80, the node is slow

Apply automatic fix

Use big data for tech use cases

Design platform for slowness management

Automate !

criteo

Garmadon

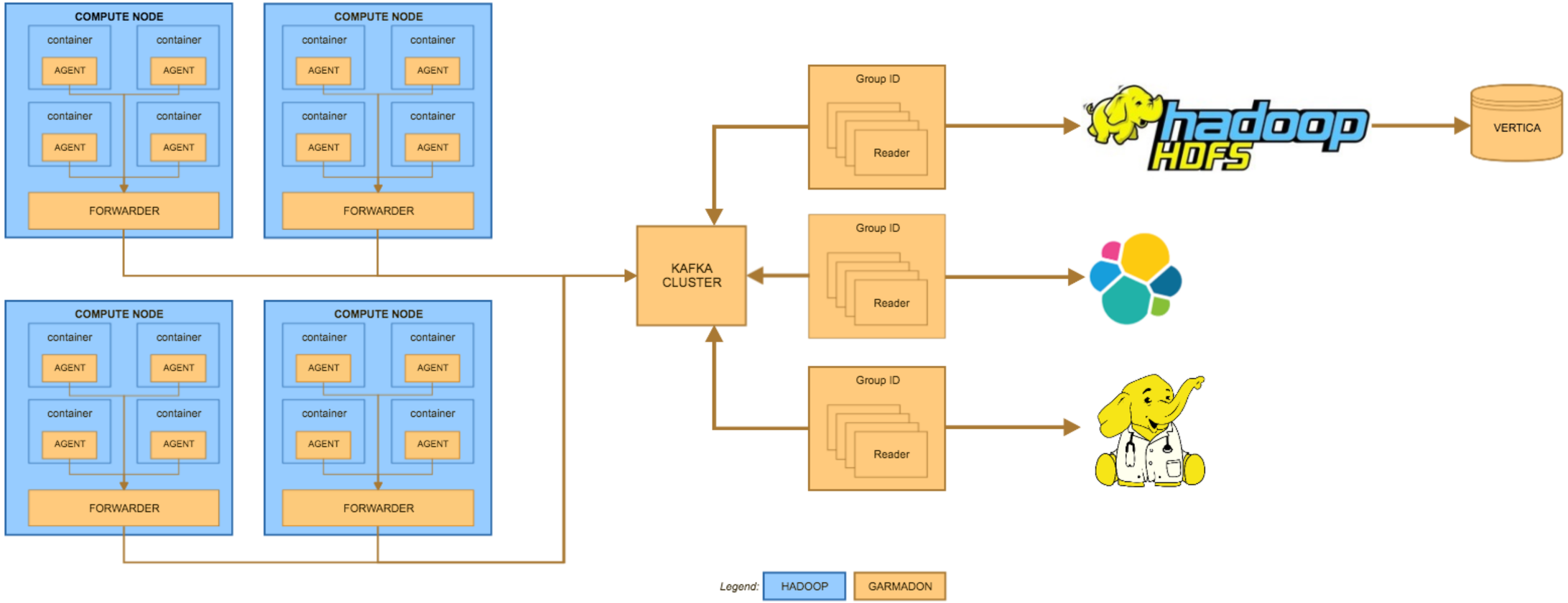
What do we need ?

Capacity planning
Managing data
Detect badly behaved users
Give feedback about containers behavior to our users
Correlate containers behavior with the platform state

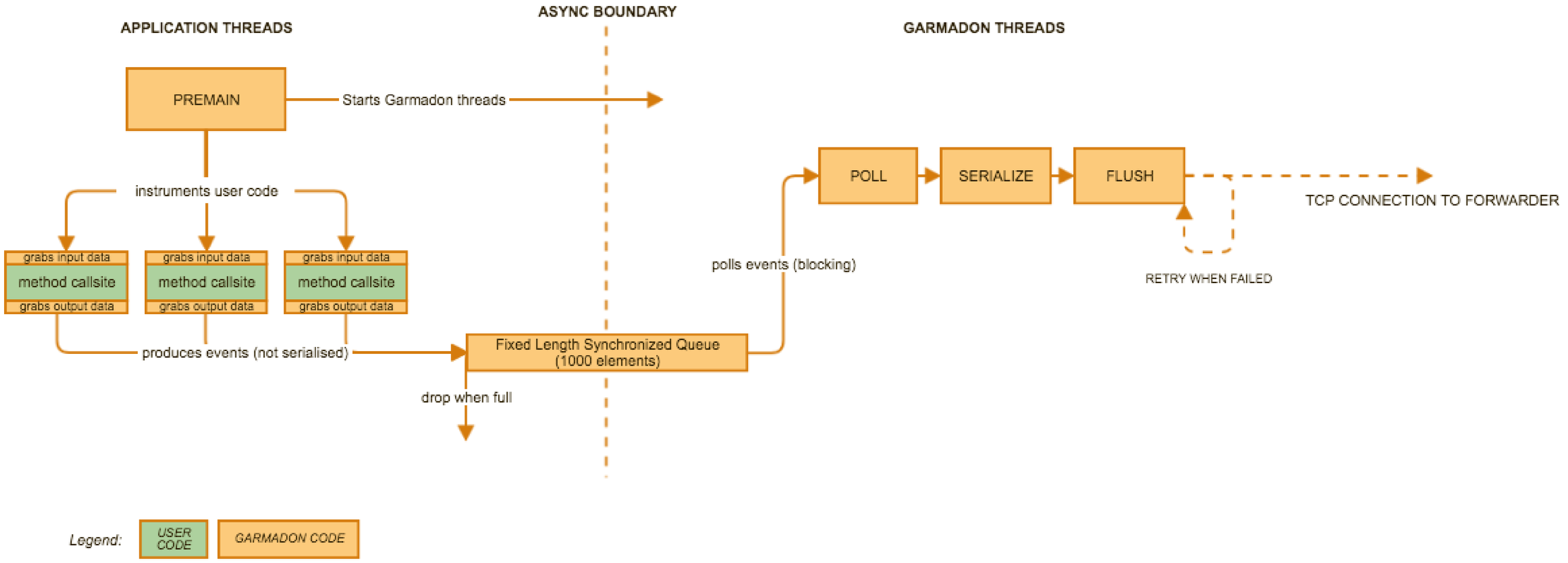
Constraints ?

Many jobs (300k per day)
Many containers (150 Million per day)
Autonomous users
Many frameworks

Garmadon – Big picture



Garmadon – Agent attachment



Garmadon – Agent attachment

Why bytebuddy is so cool ?

Nice DSL to match classes, methods and attach instrumentation code

Arguments capture via annotations

Optimized classloading lookups

Automatic code delegation strategy

Garmadon – Agent attachment

Time to instrument a container....40 sec (!)

Bytebuddy makes instrumentation easy, does the best to be fast, but does not tell you what is slow/fast

...Source code !

Garmadon – Agent attachment

Instrumentation by agent happens when code is not jitted
- ByteBuddy loading and compilation takes a bit of time, there is nothing you can do about it
- Depends on your hardware, about 400ms on our machines

Class matching via **name** is **fast** !
- String comparison

Class matching via **hierarchy** is **slow** !
- Unzipping
- Bytecode analysis

Avoid lookup on undesired classes by excluding packages
- fast because based on name
- at least, exclude ByteBuddy itself :)

Create your Agent builders and install them on parallel

criteo

Garmadon – Agent attachment

Time to instrument a container....1 sec (!)

Garmadon – Events carrying

<msg_type> (4 bytes)	<header_size> (4 bytes)	<event_size> (4 bytes)	header	event

Garmadon – Events carrying

<msg_type> (4 bytes)	<header_size> (4 bytes)	<event_size> (4 bytes)	header	event

Why Netty is so cool ?

Elegant pattern to enrich raw bytes processing

Out-of-the-box classes for framed protocols

Asynchronous and fast !

criteo

Garmadon – Numbers…

30k evt/s

1 To/day on Kafka

700 Go - 1,5 billion documents per day in Elastic Search

Know your tools

Segregate responsibilities

For simple case, simple protocol works

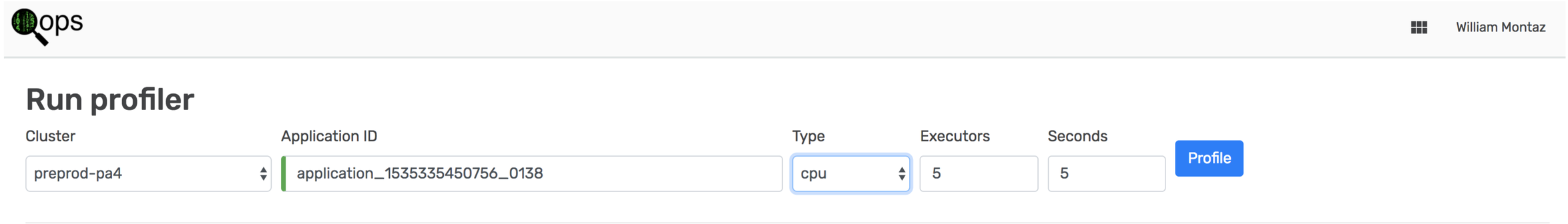
Use raw bytes

Use raw Netty

criteo

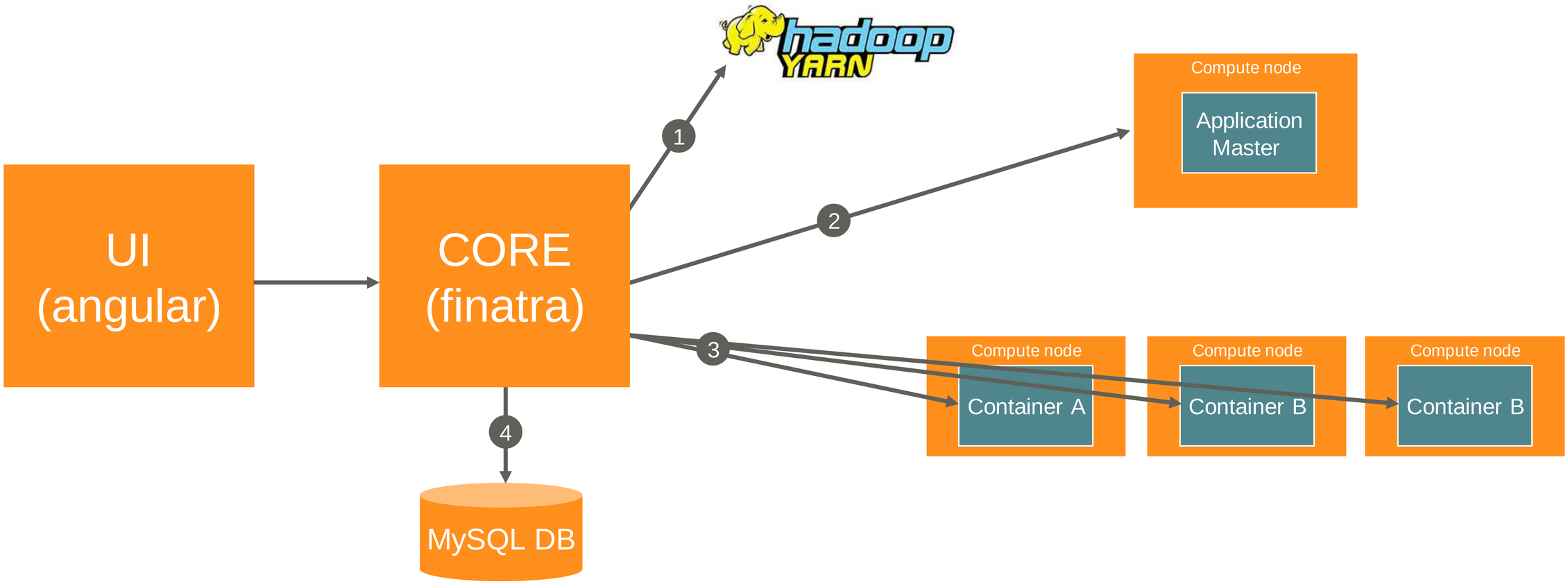
How to help user tuning their jobs

Self-Service profiling



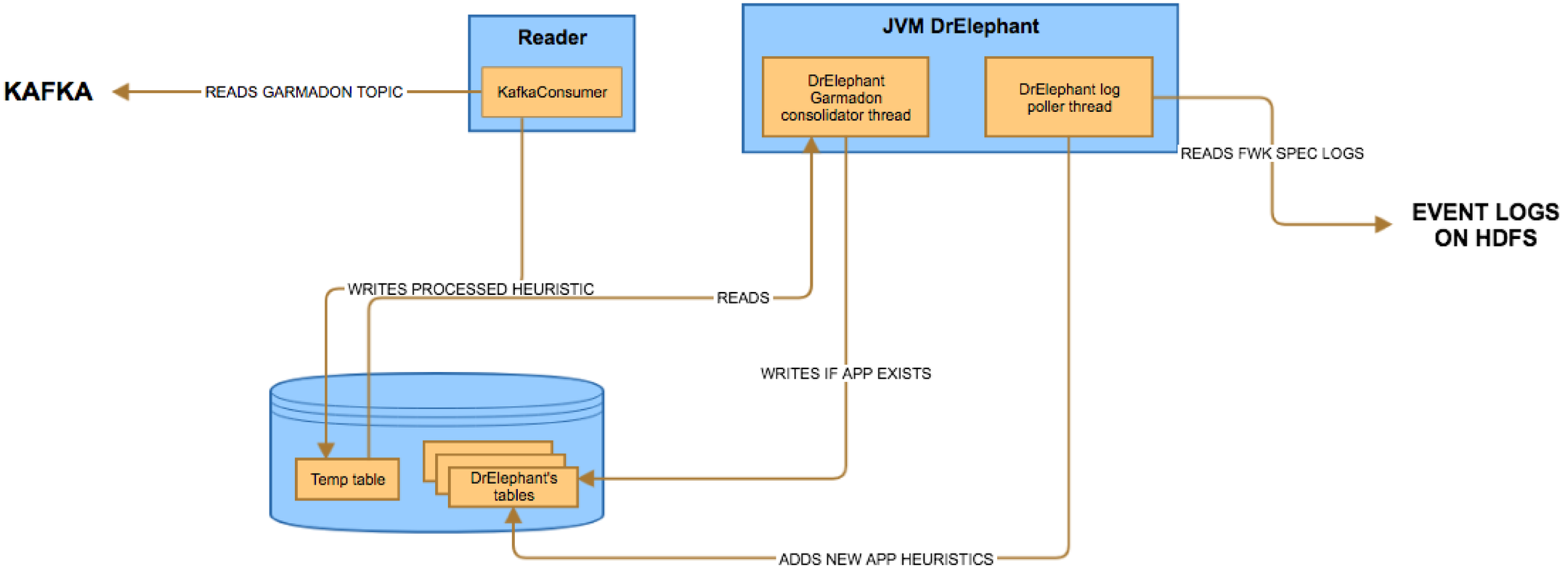
Self-Service profiling

Self-Service profiling

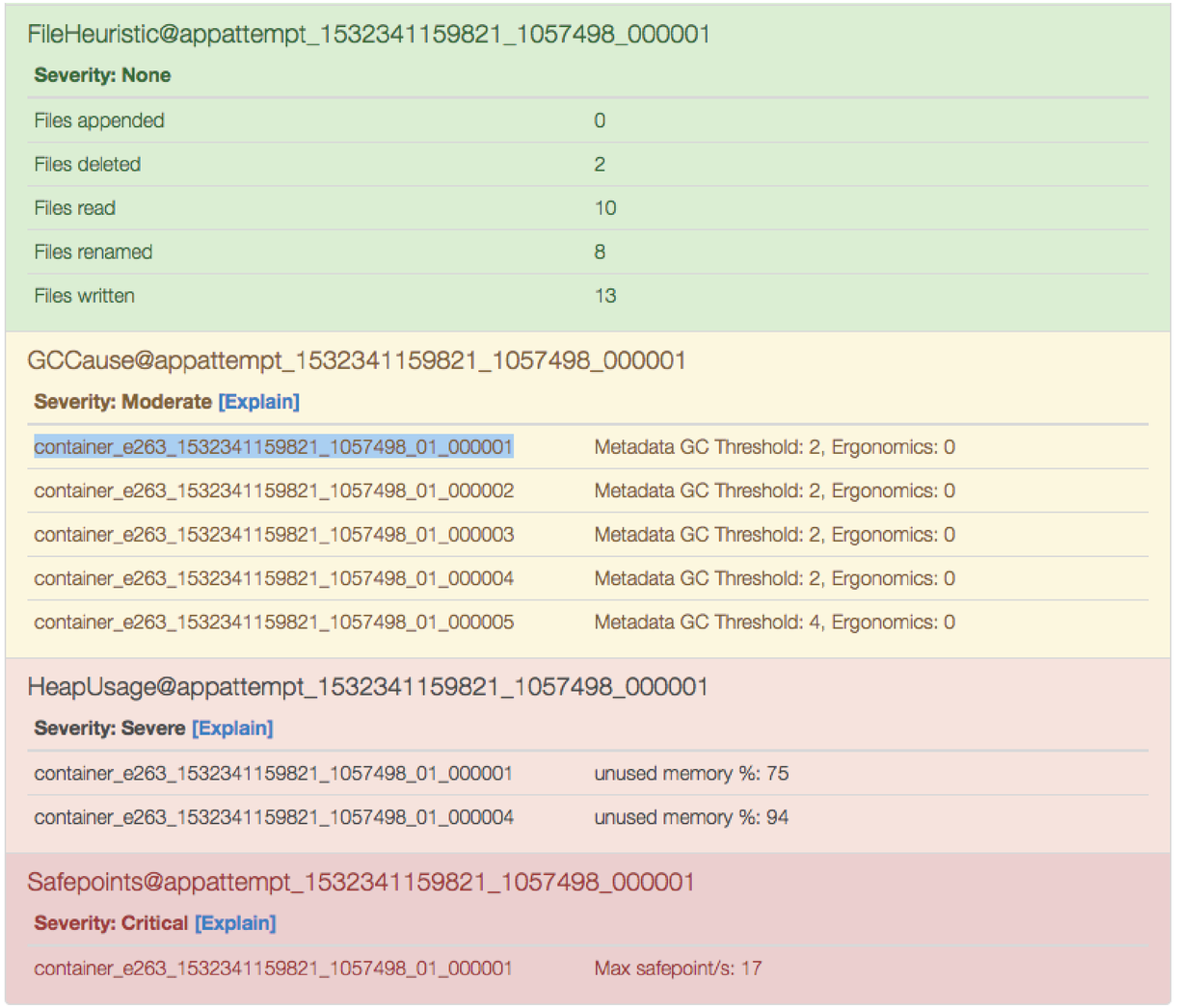


criteo

Dr-Elephant heuristics



Dr-Elephant heuristics



FileHeuristic@appattempt_1532341159821_1057498_000001

Severity: None

Files appended	0
Files deleted	2
Files read	10
Files renamed	8
Files written	13

GCCause@appattempt_1532341159821_1057498_000001

Severity: Moderate [Explain]

container_e263_1532341159821_1057498_01_000001	Metadata GC Threshold: 2, Ergonomics: 0
container_e263_1532341159821_1057498_01_000002	Metadata GC Threshold: 2, Ergonomics: 0
container_e263_1532341159821_1057498_01_000003	Metadata GC Threshold: 2, Ergonomics: 0
container_e263_1532341159821_1057498_01_000004	Metadata GC Threshold: 2, Ergonomics: 0
container_e263_1532341159821_1057498_01_000005	Metadata GC Threshold: 4, Ergonomics: 0

HeapUsage@appattempt_1532341159821_1057498_000001

Severity: Severe [Explain]

container_e263_1532341159821_1057498_01_000001	unused memory %: 75
container_e263_1532341159821_1057498_01_000004	unused memory %: 94

Safepoints@appattempt_1532341159821_1057498_000001

Severity: Critical [Explain]

container_e263_1532341159821_1057498_01_000001	Max safepoint/s: 17

criteo

Dr-Elephant heuristics

```
GarmadonReader.Builder
    .stream(kafkaConnectString)
    .withGroupId(kafkaGroupId)
    .withKafkaProp(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true")
    .withKafkaProp(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000")
    .intercept(hasTag(Header.Tag.YARN_APPLICATION).and(hasType(GarmadonSerialization.TypeMarker.GC_EVENT)), this::processGcEvent)
    .intercept(hasTag(Header.Tag.YARN_APPLICATION).and(hasType(GarmadonSerialization.TypeMarker.JVMSTATS_EVENT)), this::processJvmStatEvent)
    .intercept(hasTag(Header.Tag.YARN_APPLICATION).and(hasType(GarmadonSerialization.TypeMarker.STATE_EVENT)), this::processStateEvent)
    .intercept(
            hasTag(Header.Tag.YARN_APPLICATION).and(hasType(GarmadonSerialization.TypeMarker.FS_EVENT)),
            msg -> fileHeuristic.compute(msg.getHeader().getApplicationId(), msg.getHeader().getAppAttemptID(),
                    msg.getHeader().getContainerId(), (DataAccessEventProtos.FsEvent) msg.getBody())
    )
    .beforeIntercept(this::registerAppContainer)
    .build();
```

```
void start() { reader.startReading().whenComplete(this::completeReading); }

void stop() { reader.stopReading(); }
```


criteo

Thank you!