Trash Day: Coordinating Garbage Collection in Distributed Systems

Martin Maas* † Tim Harris† Krste Asanovic* John Kubiatowicz*

*University of California, Berkeley † Oracle Labs, Cambridge

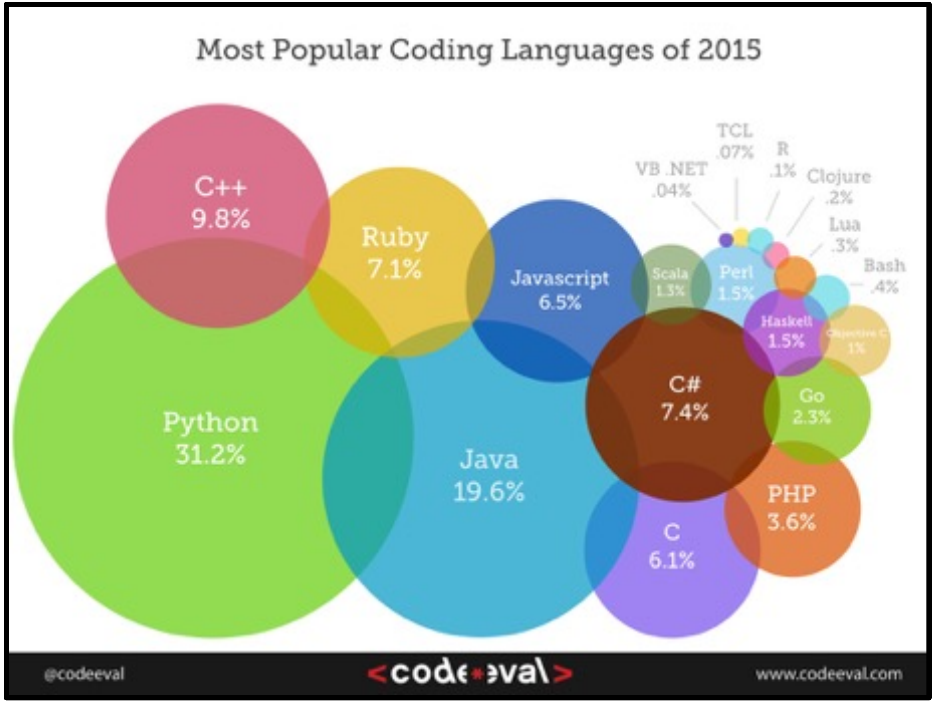
ASPIRE

ORACLE

Why you should care about GARBAGE COLLECTION in Data Center Applications

*University of California, Berkeley
† Oracle Labs, Cambridge

Most Popular Languages 2015



5 out of the top 6 languages popular in 2015 use Garbage Collection (GC)

Popular Frameworks using GC

hadoop

cassandra

AsterixDB

more engine less trunk

SHARK

HIVE

Spark

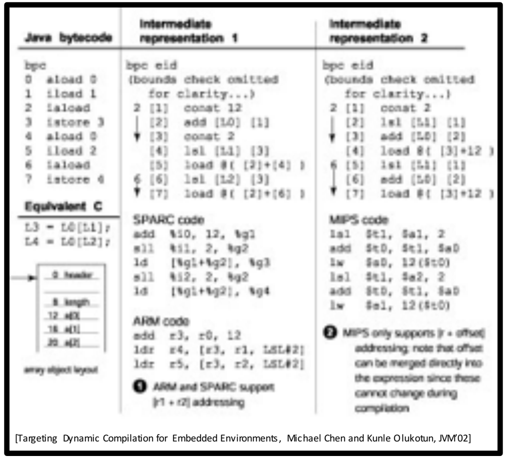
mahout

GC used by Cloud Companies

Why Managed Languages?

Productivity Gains

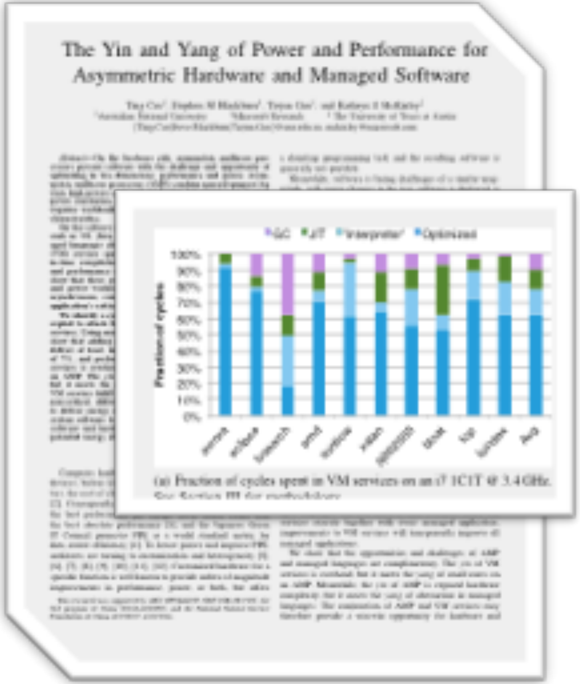
Avoiding Bugs



[Targeting Dynamic Compilation for Embedded Environments, Michael Chen and Kunle Olukotun, JVM'02]

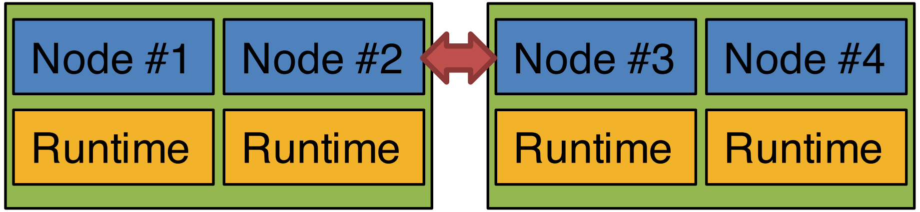
Enable Certain Optimizations

What is the Cost of GC?


The Yin and Yang of Power and Performance for Asymmetric Hardware and Managed Software

ISCA’12: Cao et al.

- GC overhead workload and heap-size dependent, 5-20% on single machine
- In Distributed Applications, additional overheads emerge. Applications run a-cross independent runtime systems:

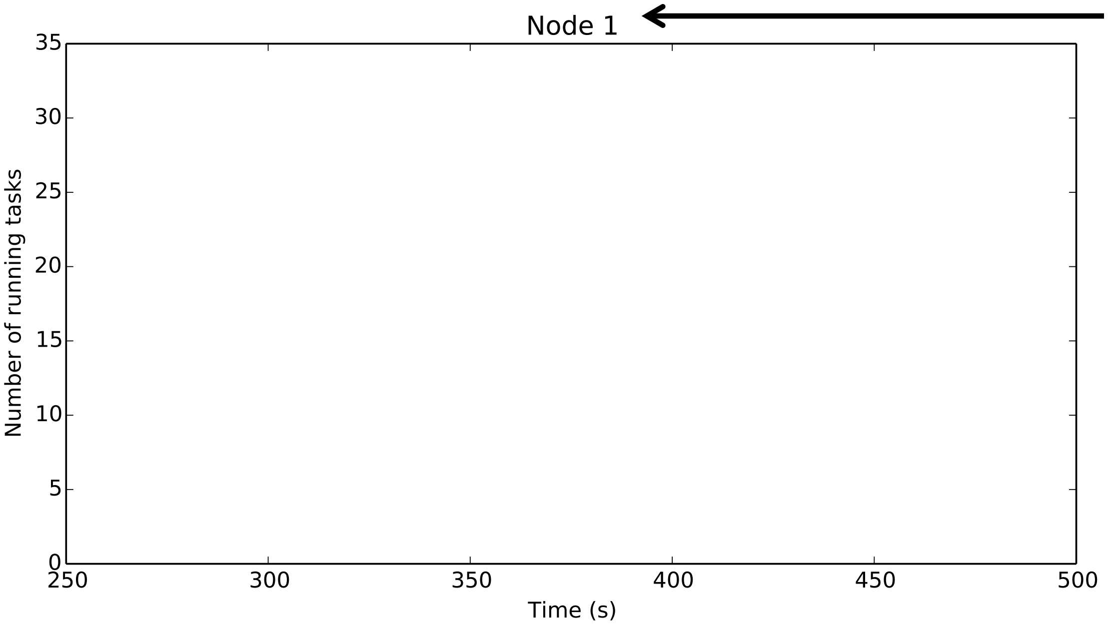


Two Example Workloads

Throughput-oriented
Batch-style

Latency-sensitive
Interactive

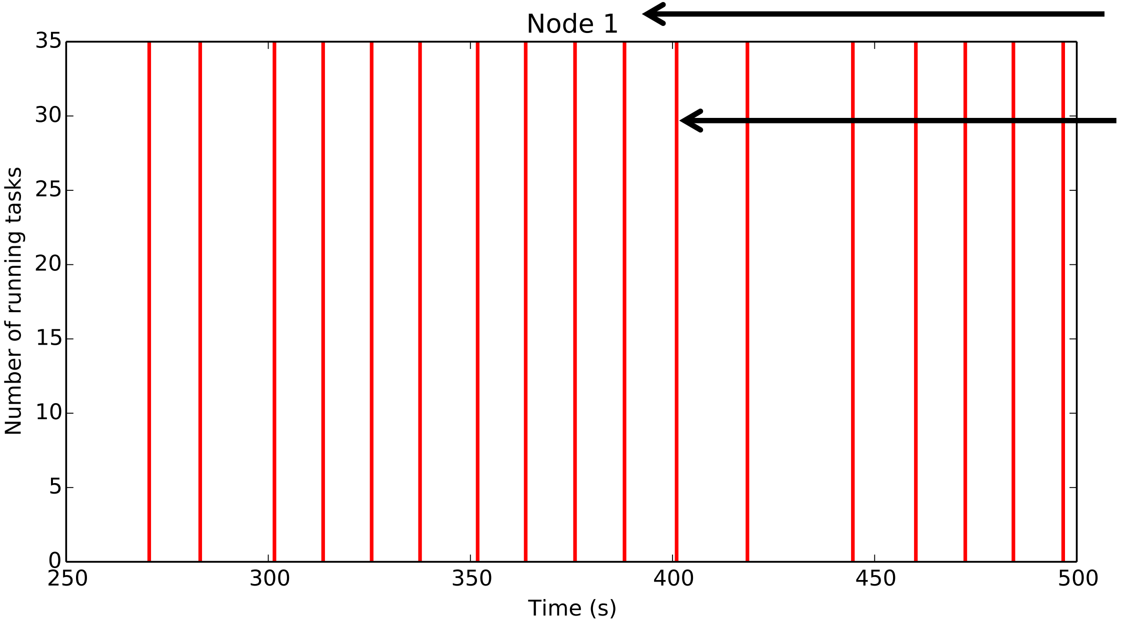
Spark Running PageRank



8-node cluster

PageRank on 56 GB Wikipedia web graph

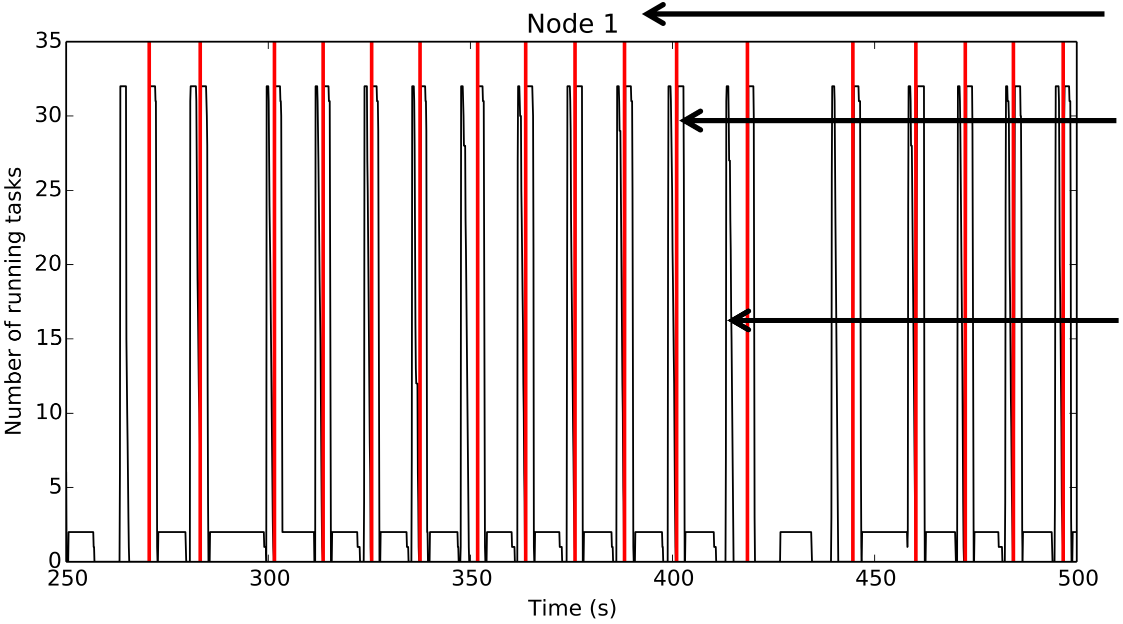
Spark Running PageRank



8-node cluster

Execution is divided into supersteps

Spark Running PageRank

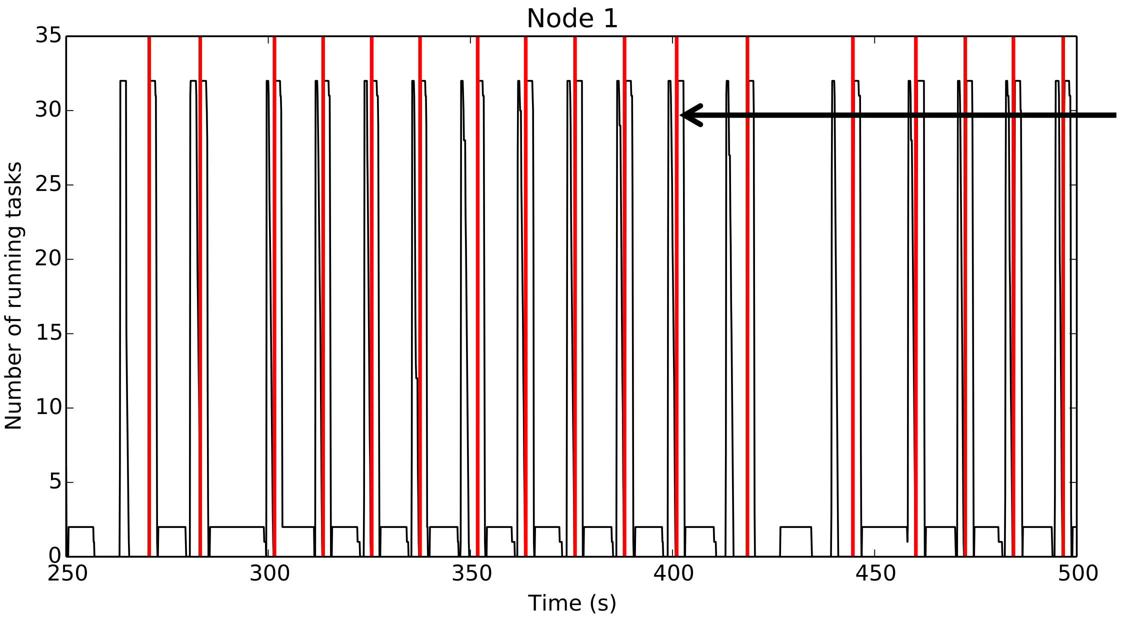


8-node cluster

Execution is divided into supersteps

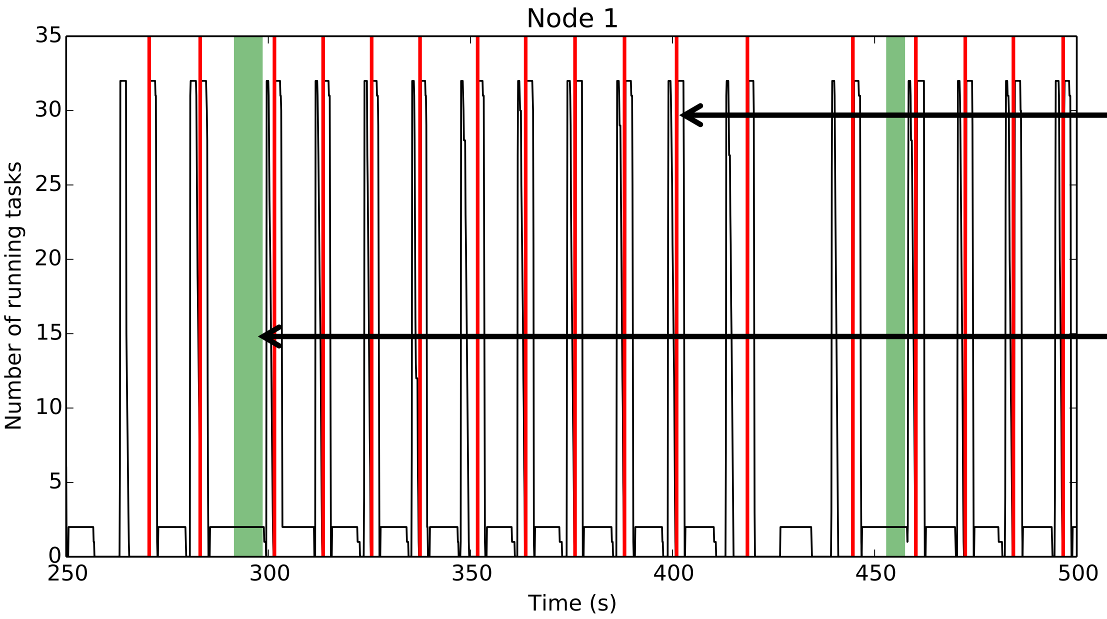
Each superstep runs independent tasks

Spark Running PageRank



Red – Synchronization at end of superstep

Spark Running PageRank

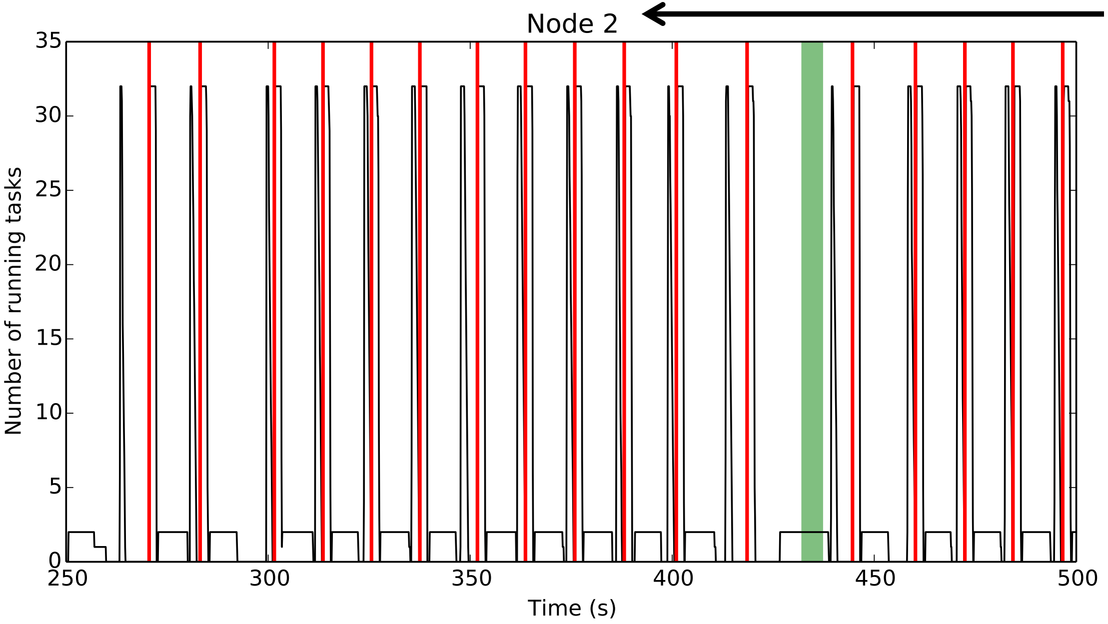


Red – Synchronization at end of superstep

Green – Major GC Pause

GC prevents superstep from completing

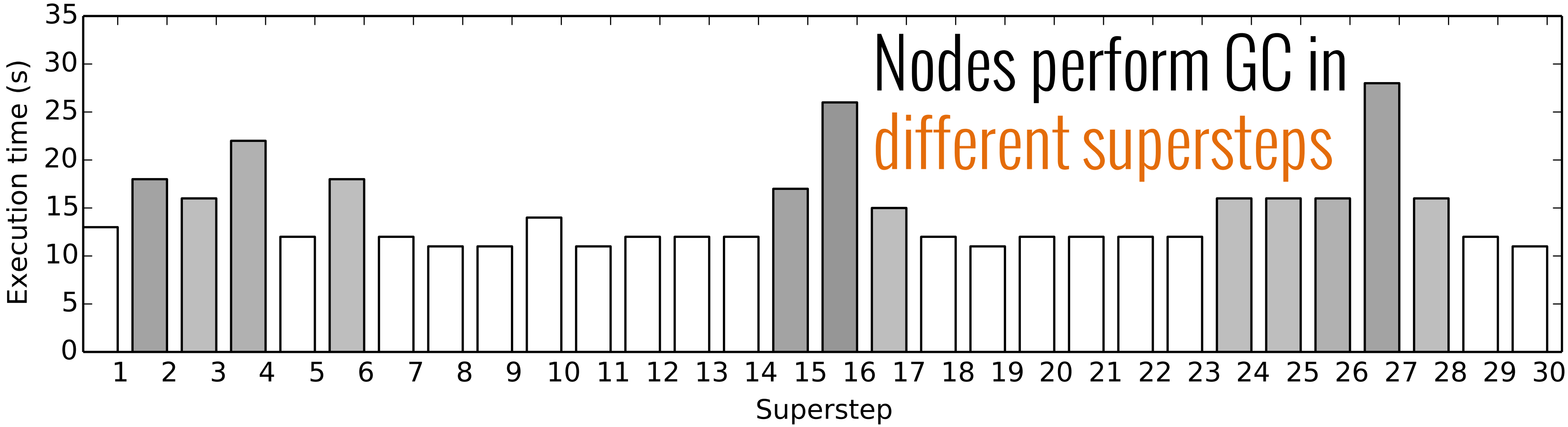
Spark Running PageRank



Different node

Execution stalls due to GC on other node

Impact on Superstep Times



White = No GC during superstep

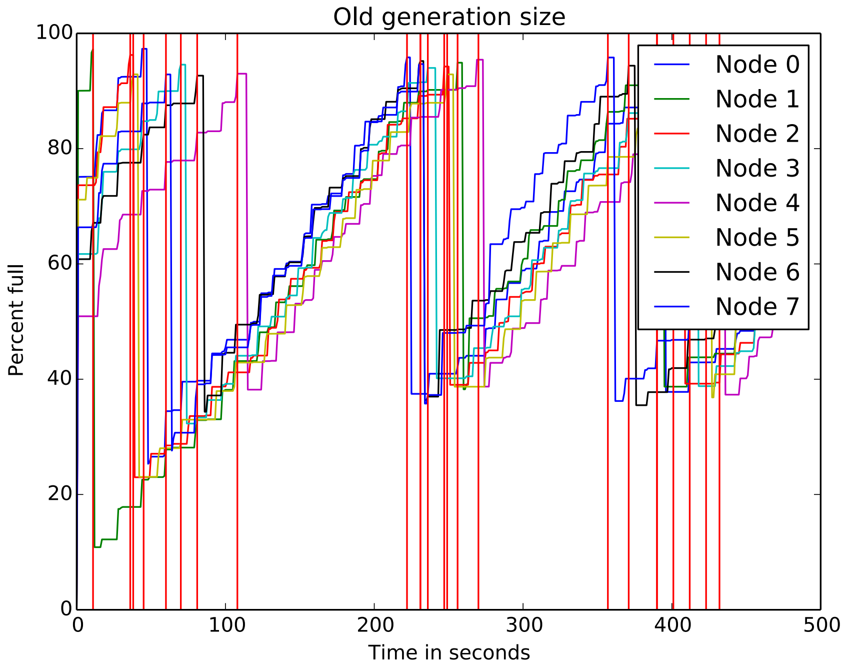
Dark = One or more GCs (the darker the more GCs)

Idea: Coordinate GC on different nodes

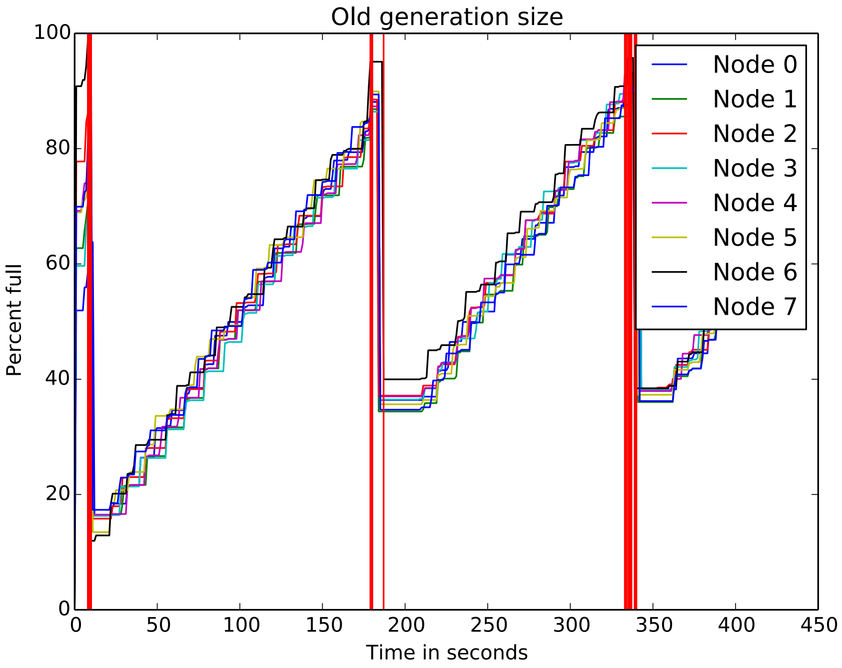
Trigger collection on all nodes at the when any one reaches a threshold

Policy: Stop-the-world Everywhere, STWE

Memory Occupancy over Time

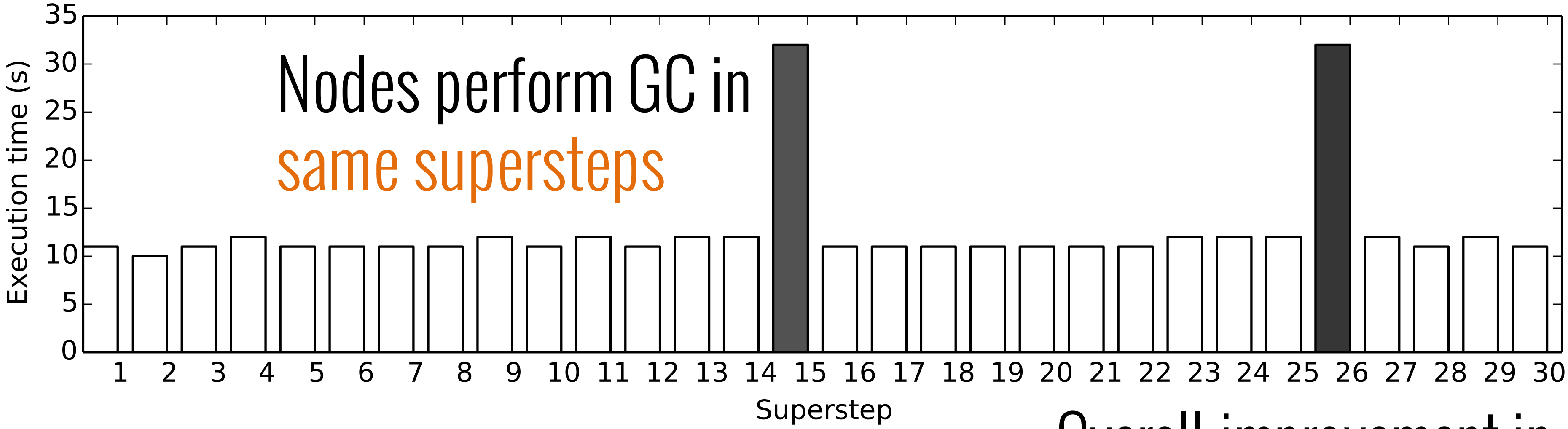


Without STWE



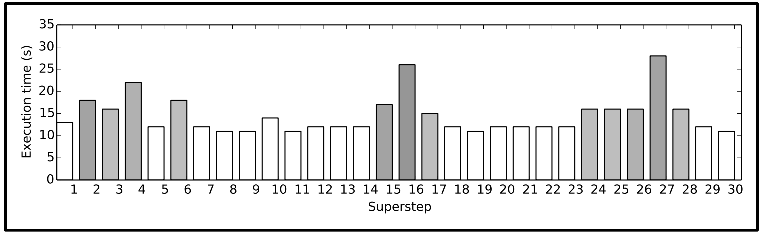
With STWE

Impact of STWE Policy



Nodes perform GC in same supersteps

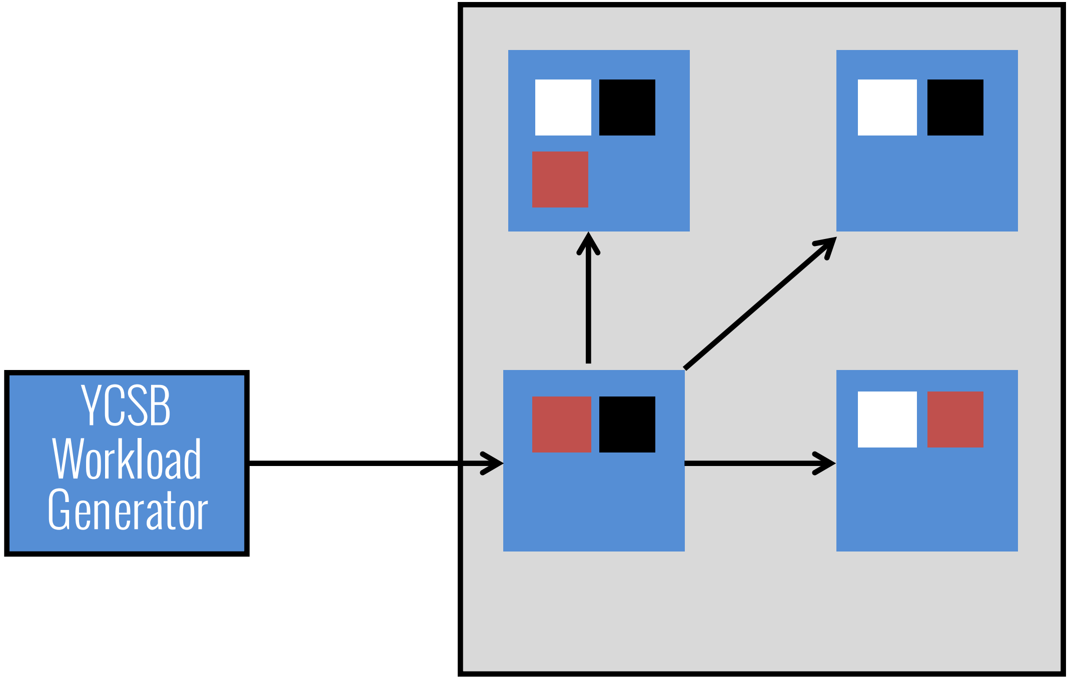
Overall improvement in execution time (~15%)



Before

cassandra

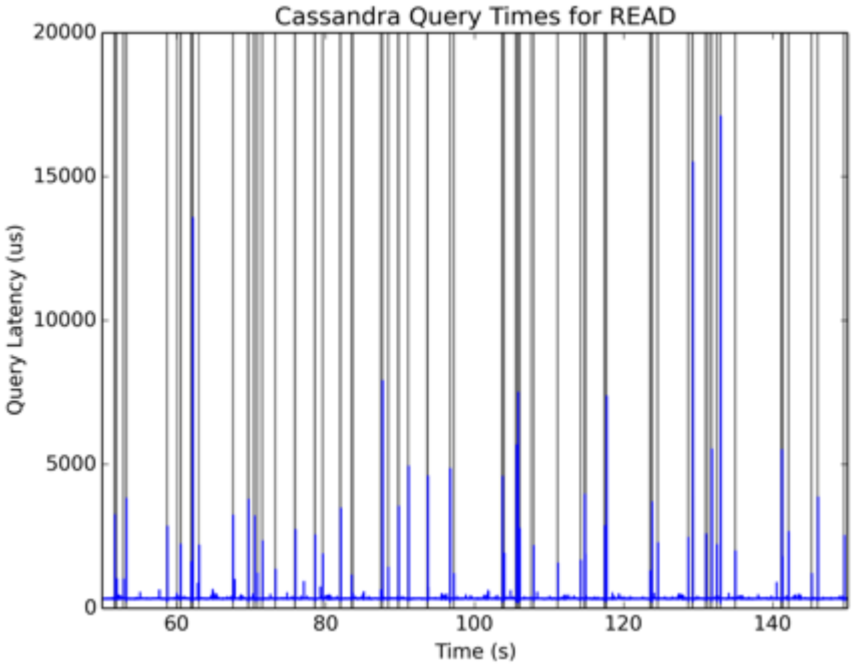
Cassandra with YCSB



4-node Cassandra Cluster
3-way replicated

Requests sent to arbitrary node; becomes coordinator and contacts replicas to assemble quorum.

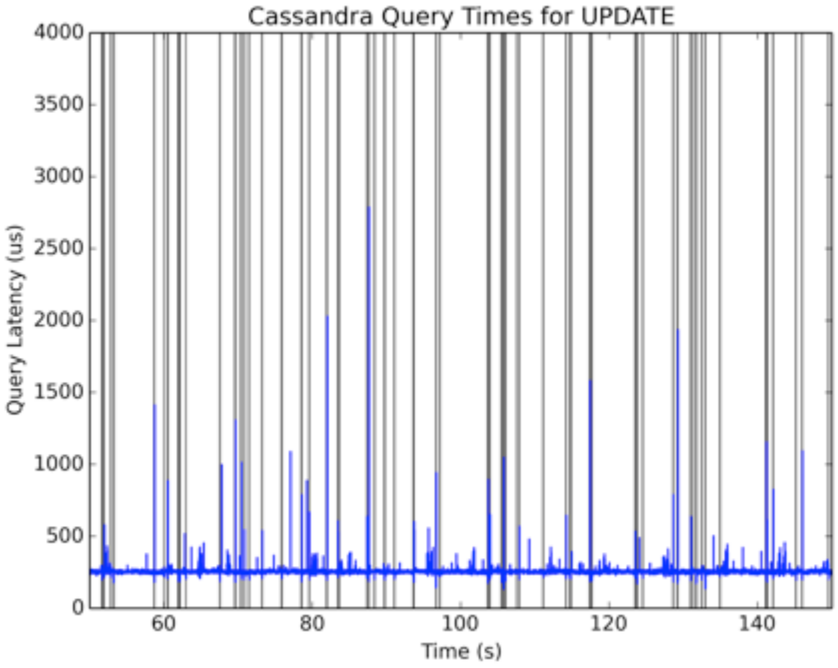
Query Latencies over Time



Blue – mean latency over a 10ms window

Grey Bars – minor GC on any node in the cluster

Query Latencies over Time



Blue – mean latency over a 10ms window

Grey Bars – minor GC on any node in the cluster

Sources of Stragglers

1. Coordinator incurs GC during request
2. Node required a quorum incurs GC
3. Non-GC reasons (e.g., anti-entropy)

Sources of Stragglers

1. Coordinator incurs GC during request
2. Node required a quorum incurs GC
3. Non-GC reasons (e.g., anti-entropy)

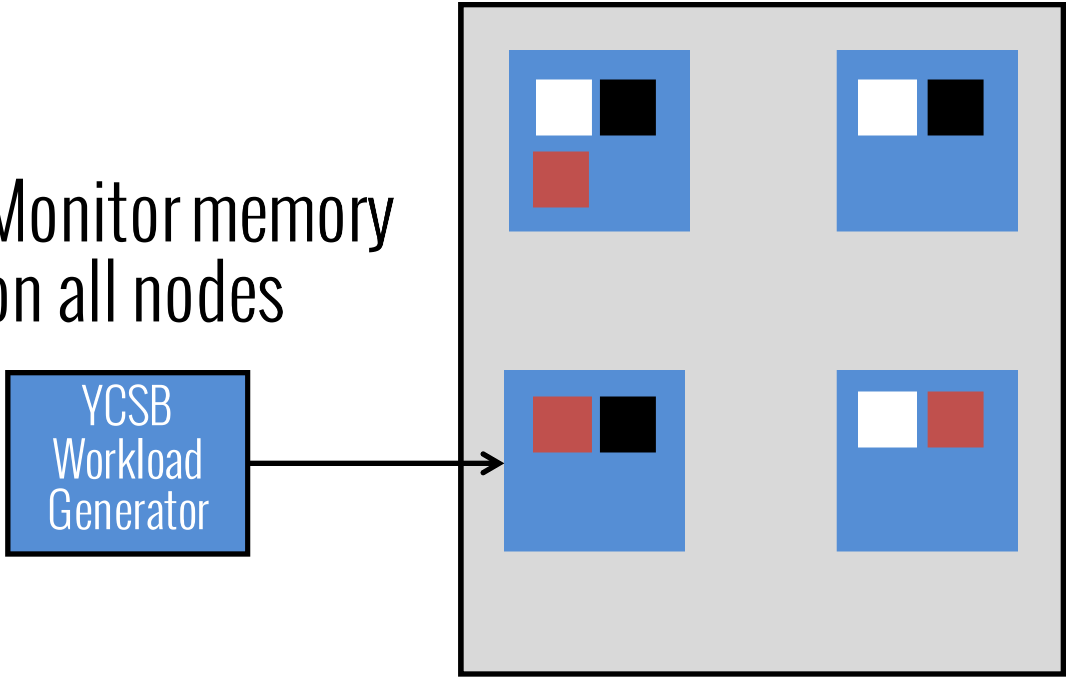
GC-aware Work Distribution

Steer client requests to Cassandra nodes, avoiding ones that will need a minor collection soon

Policy: Request Steering, STEER

Steering Cassandra Requests

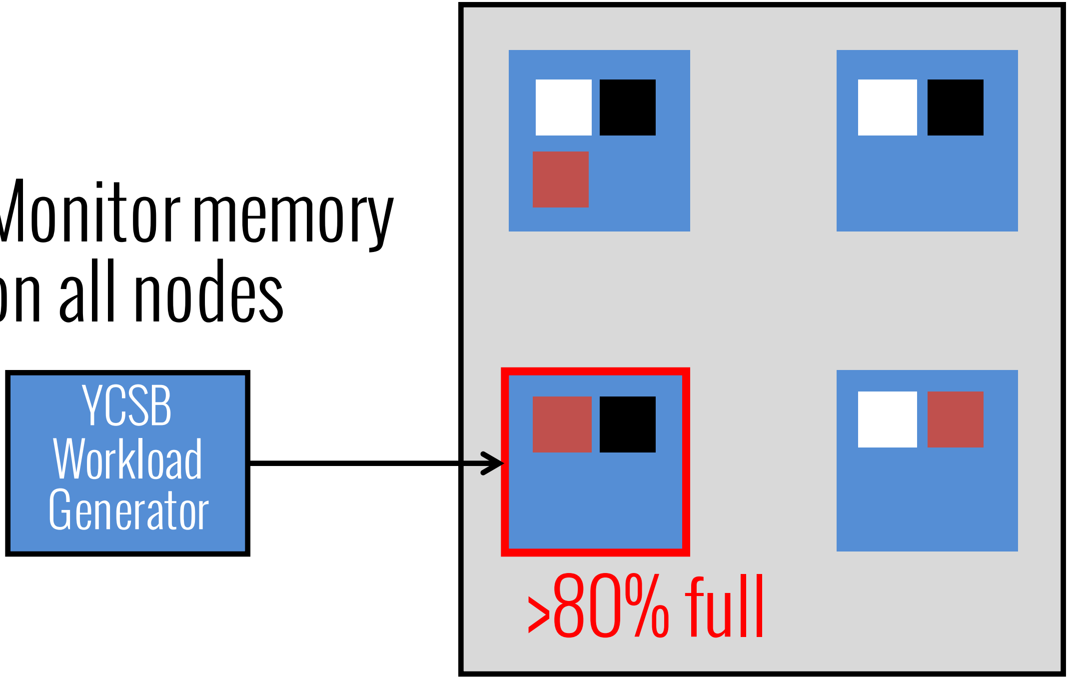
Monitor memory
on all nodes



If one node is close to GC,
send to other nodes instead

Steering Cassandra Requests

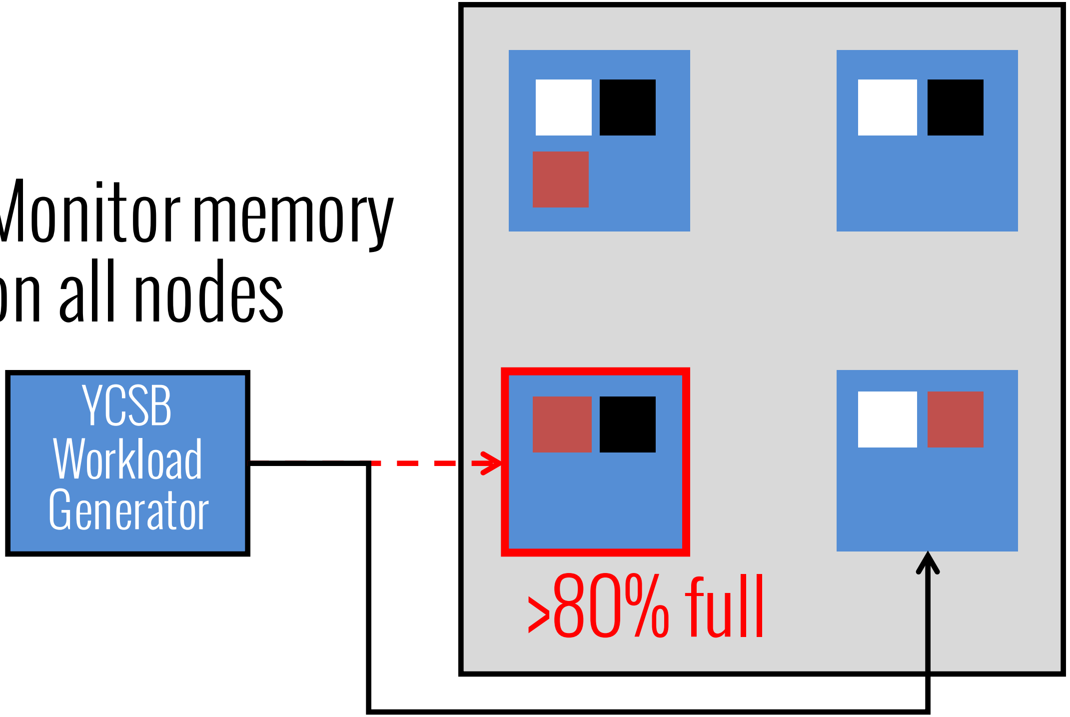
Monitor memory
on all nodes



If one node is close to GC,
send to other nodes instead

Steering Cassandra Requests

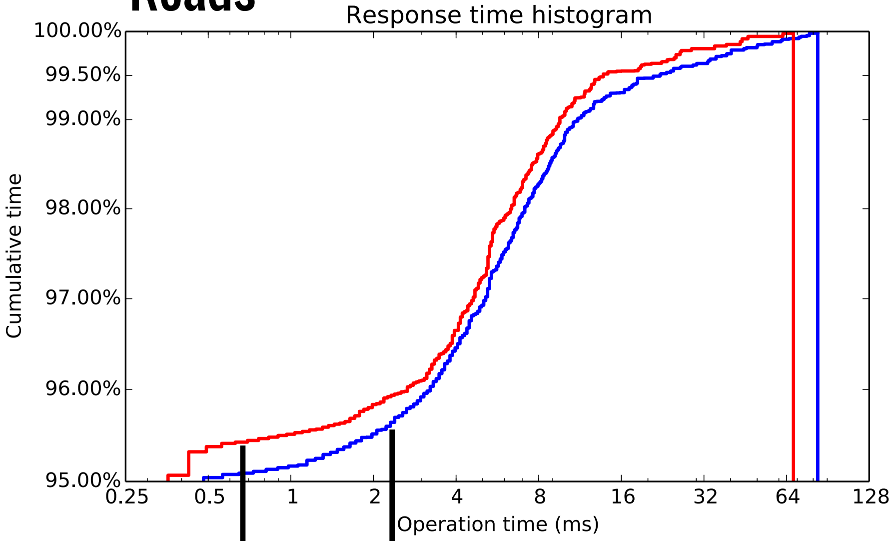
Monitor memory on all nodes



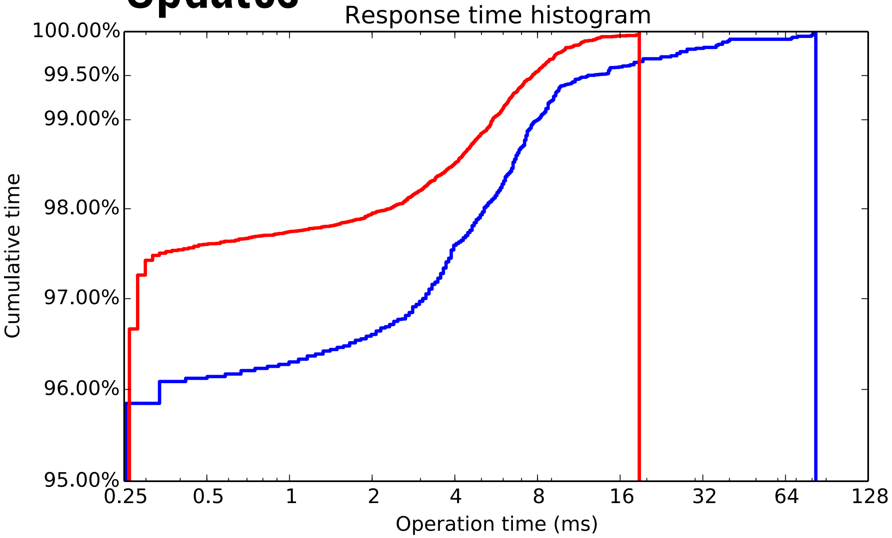
If one node is close to GC, send to other nodes instead

Impact of Request Steering

Reads



Updates

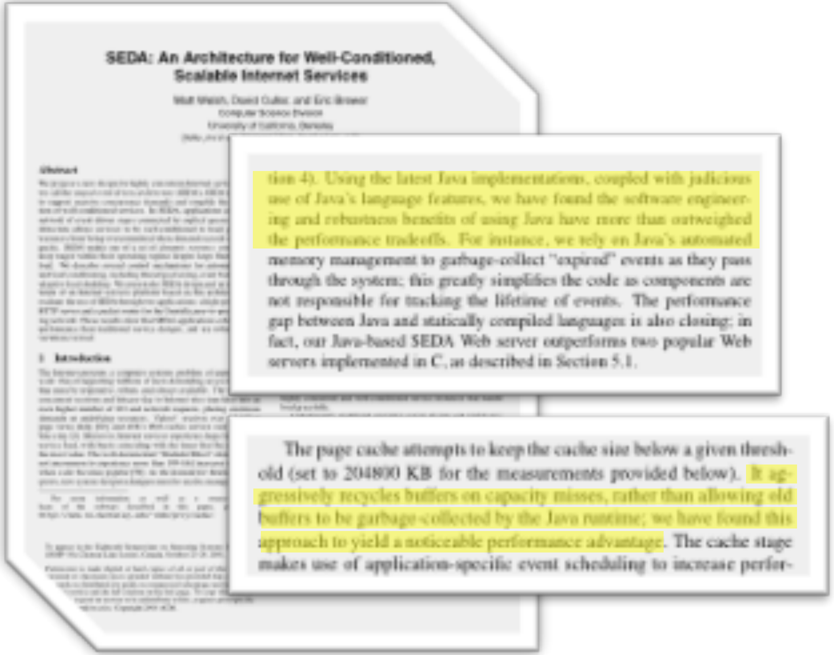


Blue – without steering

Red – with steering

99.9 percentile:	3.3 ms -> 1.6 ms
Worst case:	83 ms -> 19 ms

Are These Problems Common?


SEDA: An Architecture for Well-Conditioned, Scalable Internet Services

tion 4). Using the latest Java implementations, coupled with judicious use of Java's language features, we have found the software engineering and robustness benefits of using Java have more than outweighed the performance tradeoffs. For instance, we rely on Java's automated memory management to garbage-collect "expired" events as they pass through the system; this greatly simplifies the code as components are not responsible for tracking the lifetime of events. The performance gap between Java and statically compiled languages is also closing; in fact, our Java-based SEDA Web server outperforms two popular Web servers implemented in C, as described in Section 5.1.

The page cache attempts to keep the cache size below a given threshold (set to 204800 KB for the measurements provided below). It aggressively recycles buffers on capacity misses, rather than allowing old buffers to be garbage-collected by the Java runtime; we have found this approach to yield a noticeable performance advantage. The cache stage makes use of application-specific event scheduling to increase perfor-

SOSP '01, Welsh et al.

- GC problems affect a large number of applications
- Have existed since dawn of warehouse-scale computing
- Current surge of interest in both industry and academia (6 new papers in last 4 mo.)

Common Solutions

✘ Lose language advantages, lack of generality

Rewrite at lower level

✘ Substantial effort to adopt

Respond to GC Pauses

✘ Performance overheads, still have pauses

Concurrent Collectors

Common Solutions

✗ Lose language advantages

✗ Substantial effort to adopt

✗ Performance Overheads

No general, widely adopted solution!

Rewrite at lower level

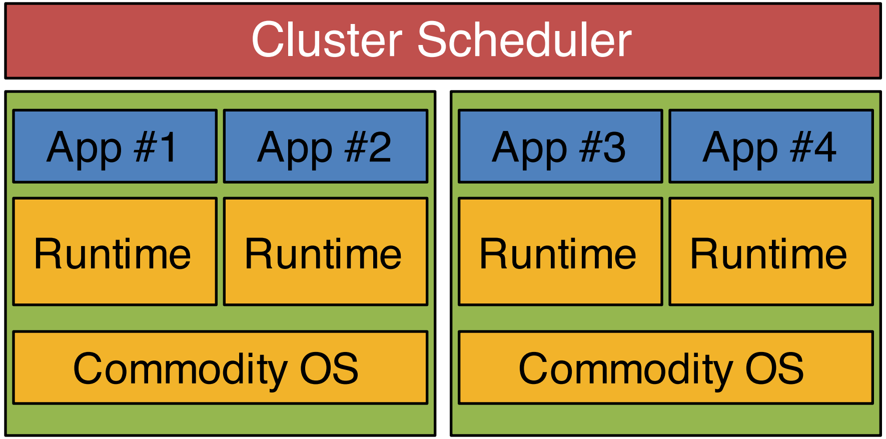
Respond to GC Pauses

Concurrent collectors

The problem is not GC, it is language runtime system coordination

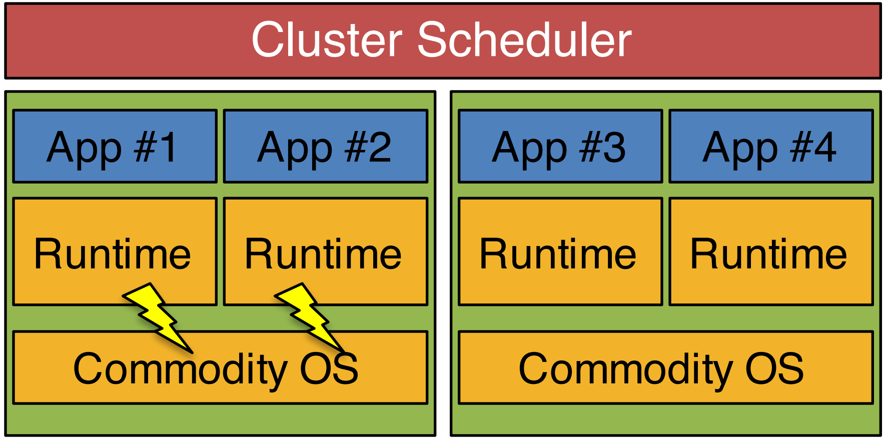
Current Approach

Language Runtime Systems are completely independent (not just GC)



Current Approach

Language Runtime Systems are completely independent (not just GC)

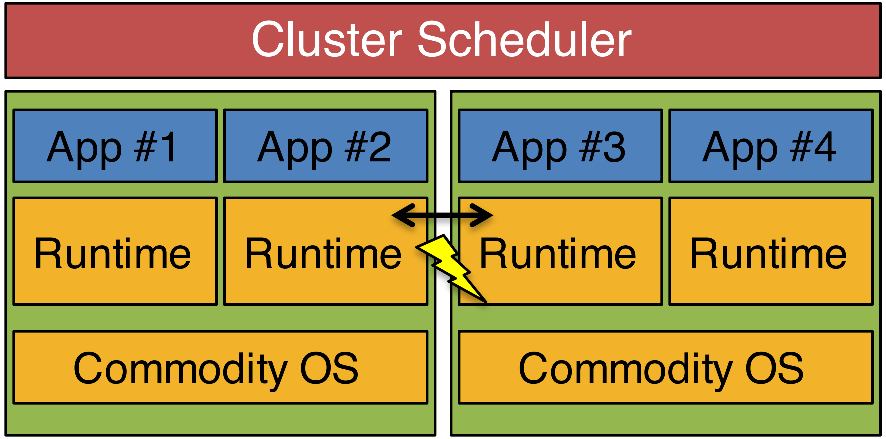


Intra-node Interference

Current Approach

Language Runtime Systems are completely independent (not just GC)

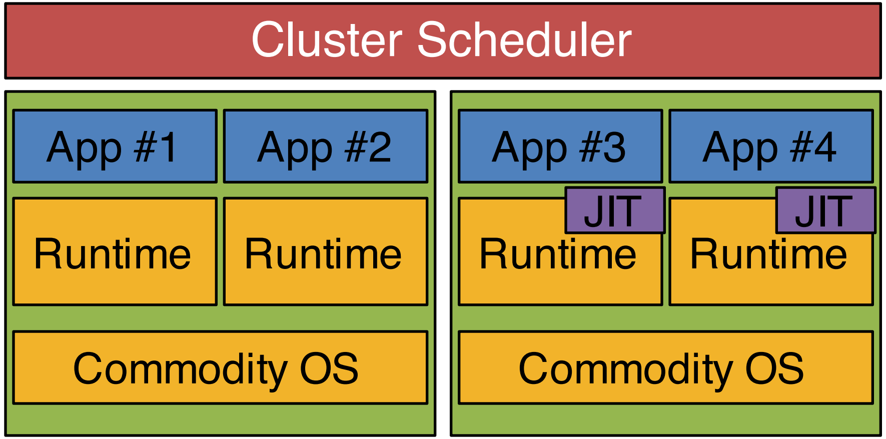
Intra-node Interference



Lack of Coordination

Current Approach

Language Runtime Systems are completely independent (not just GC)



Redundancy

Intra-node Interference

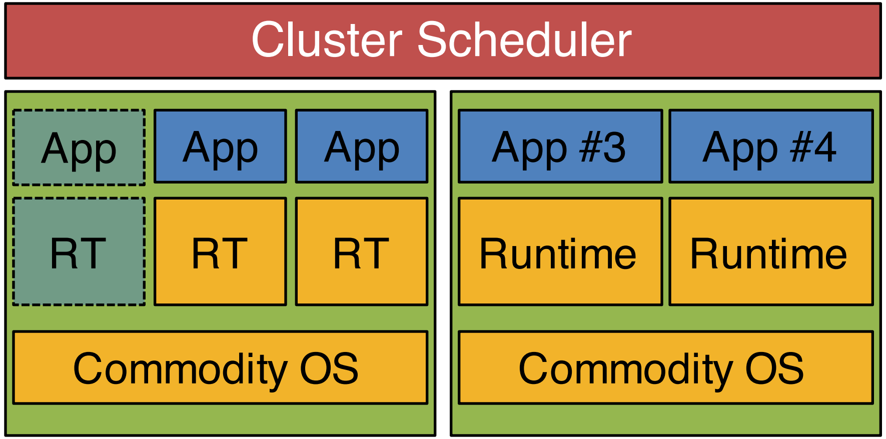
Lack of Coordination

Current Approach

Language Runtime Systems are completely independent (not just GC)

Elasticity

Intra-node Interference

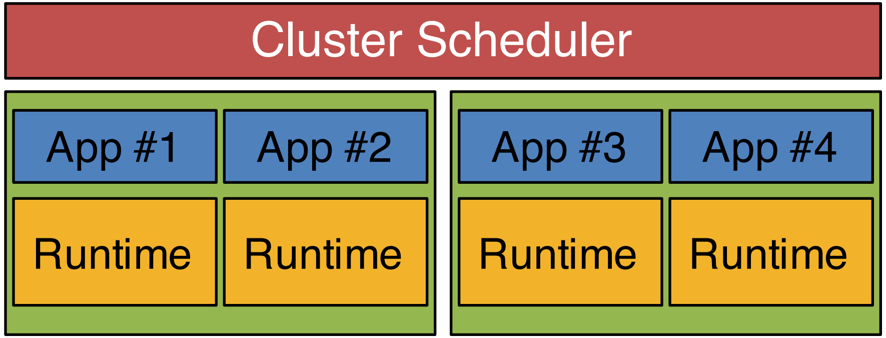


Redundancy

Lack of Coordination

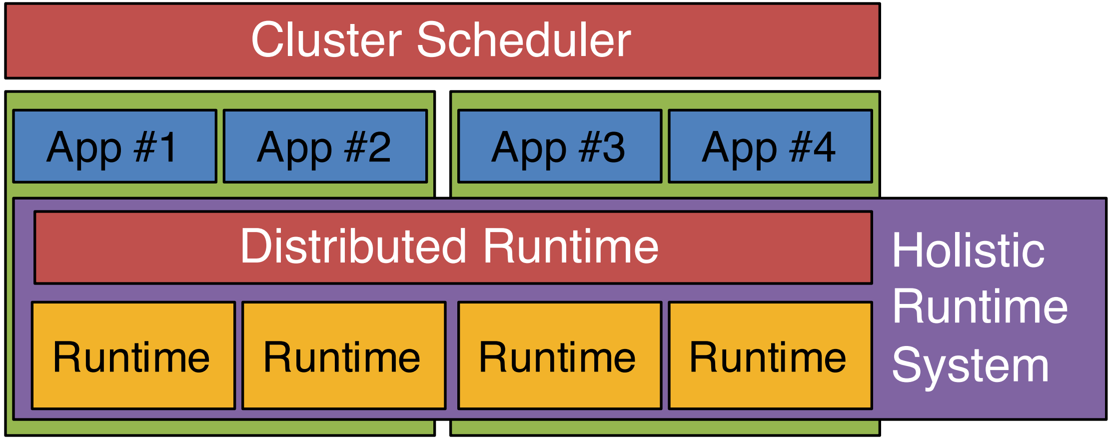
Holistic Runtime Systems

Apply the Distributed OS Ideas to design a Distributed Language Runtime System



Holistic Runtime Systems

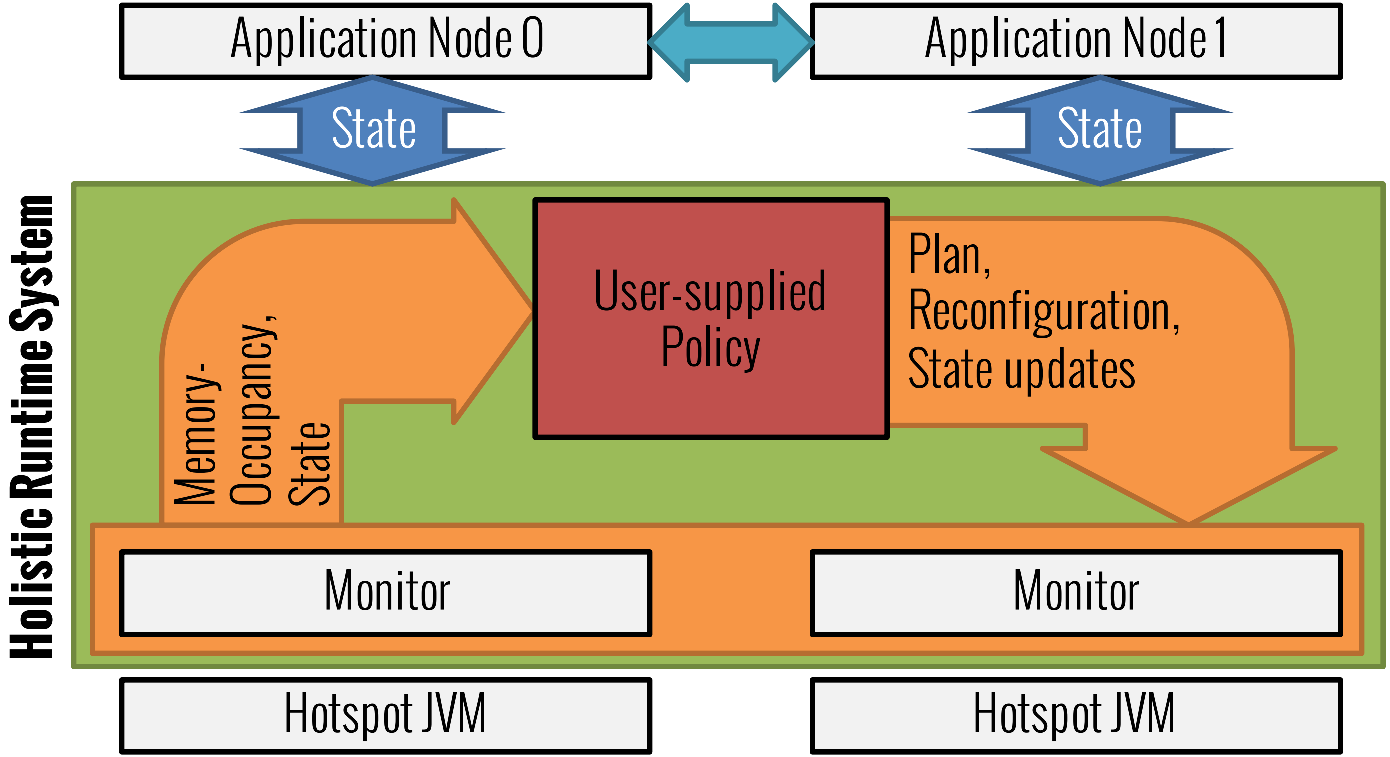
Apply the Distributed OS Ideas to design a Distributed Language Runtime System



Our Prototype

- Coordinated runtime decisions using a feedback loop with dist. consensus
- Configured by Policy (written in DSL)
- Drop-in replacement for Java VM
- No modifying of application required

System Design



Why This Approach?

- Easy to adopt (just pick policy, almost no configuration required)
- Minimally invasive to runtime system
- Expressive (can express a large range of GC coordination policies)

Our plan is to make the system available as open source

Would you use it?

Thank you! Any Questions?

Martin Maas, Tim Harris, Krste Asanovic, John Kubiatowicz

{maas,krste,kubitron}@eecs.berkeley.edu timothy.l.harris@oracle.com

Work started while at Oracle Labs, Cambridge.