Trash Day: Coordinating Garbage Collection in Distributed Systems

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

*University of California, Berkeley  † Oracle Labs, Cambridge
Why you should care about GARBAGE COLLECTION in Data Center Applications
Most Popular Languages 2015

5 out of the top 6 languages popular in 2015 use Garbage Collection (GC)
Popular Frameworks using GC
GC used by Cloud Companies

- .NET
- Python
- JavaScript
- Scala
- Java
- PHP
- Ruby
- Google
Why Managed Languages?

Productivity Gains

Avoiding Bugs

Enable Certain Optimizations

Targeting Dynamic Compilation for Embedded Environments, Michael Chen and Kunle Olukotun, JVM'02
What is the Cost of GC?

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

```
<table>
<thead>
<tr>
<th>Node #1</th>
<th>Node #2</th>
<th>Node #3</th>
<th>Node #4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
</tr>
</tbody>
</table>
```

ISCA’12: Cao et al.
Two Example Workloads

Throughput-oriented
Batch-style

Latency-sensitive
Interactive

Spark

Cassandra
Spark Running PageRank

8-node cluster

PageRank on 56 GB Wikipedia web graph
Spark Running PageRank

Execution is divided into supersteps

8-node cluster
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

Green – Major GC Pause
Red – Synchronization at end of superstep

GC prevents superstep from completing
Spark Running PageRank

Execution stalls due to GC on other node

Different 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

![Graph showing memory occupancy over time without STWE.]

With STWE

![Graph showing memory occupancy over time with STWE.]

Percent full

Time in seconds

Old generation size

Node 0

Node 1

Node 2

Node 3

Node 4

Node 5

Node 6

Node 7
Impact of STWE Policy

Nodes perform GC in same supersteps

Overall improvement in execution time (~15%)
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.

YCSB Workload Generator

>80% full
Monitor memory on all nodes

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

Steering Cassandra Requests
Impact of Request Steering

Reads

Response time histogram

- 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?

- 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.)

SOSP ’01, Welsh et al.
Common Solutions

- **Rewrite at lower level**
  - Lose language advantages, lack of generality

- **Respond to GC Pauses**
  - Substantial effort to adopt

- **Concurrent Collectors**
  - Performance overheads, still have pauses
No general, widely adopted solution!
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)

Intra-node Interference

Redundancy

Lack of Coordination
Current Approach

Language Runtime Systems are completely independent (not just GC)

Elasticity

Intra-node Interference

Cluster Scheduler

<table>
<thead>
<tr>
<th>App</th>
<th>App</th>
<th>App</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT</td>
<td>RT</td>
<td>RT</td>
</tr>
<tr>
<td>Commodity OS</td>
<td>Commodity OS</td>
<td></td>
</tr>
</tbody>
</table>

App #3 | App #4

Runtime | Runtime

Redundancy

Lack of Coordination
Holistic Runtime Systems

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

Cluster Scheduler

App #1  App #2
Runtime  Runtime

App #3  App #4
Runtime  Runtime
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

Application Node 0

State

Application Node 1

State

User-supplied Policy

Plan, Reconfiguration, State updates

Memory, Occupancy, State

Monitor

Hotspot JVM

Holistic Runtime System

Monitor

Hotspot JVM
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.