ALMA - GC-assisted JVM Live Migration for Java Server Applications

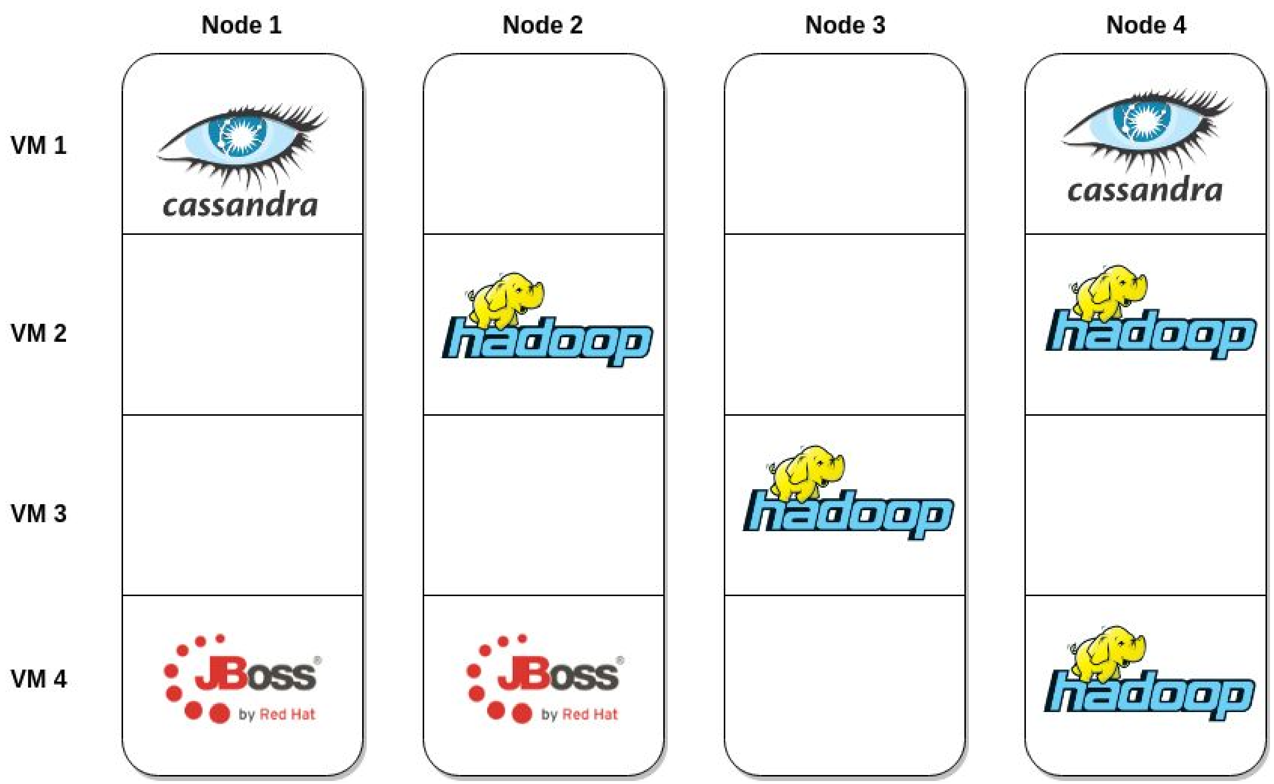
Rodrigo Bruno, Paulo Ferreira

{rodrigo.bruno,paulo.ferreira}@inesc-id.pt

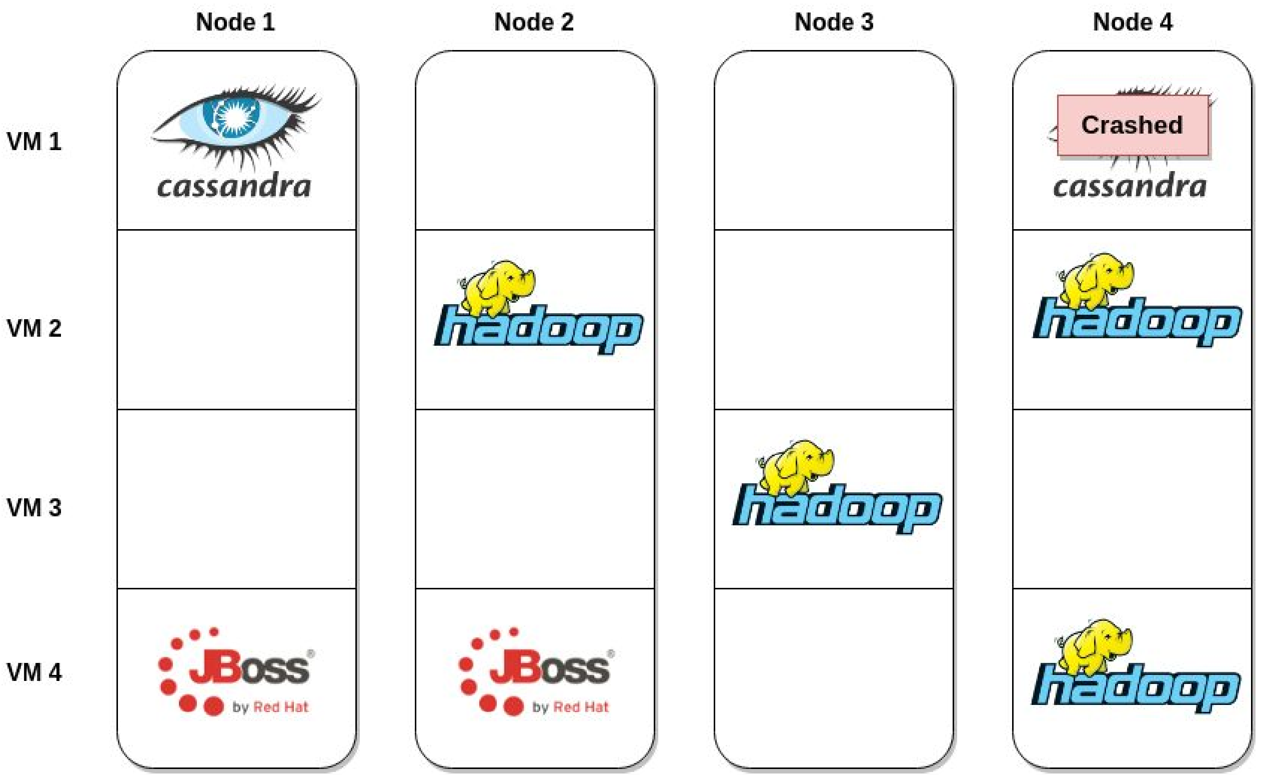
INESC-ID - Instituto Superior Técnico, ULisboa

Middleware’16@Trento

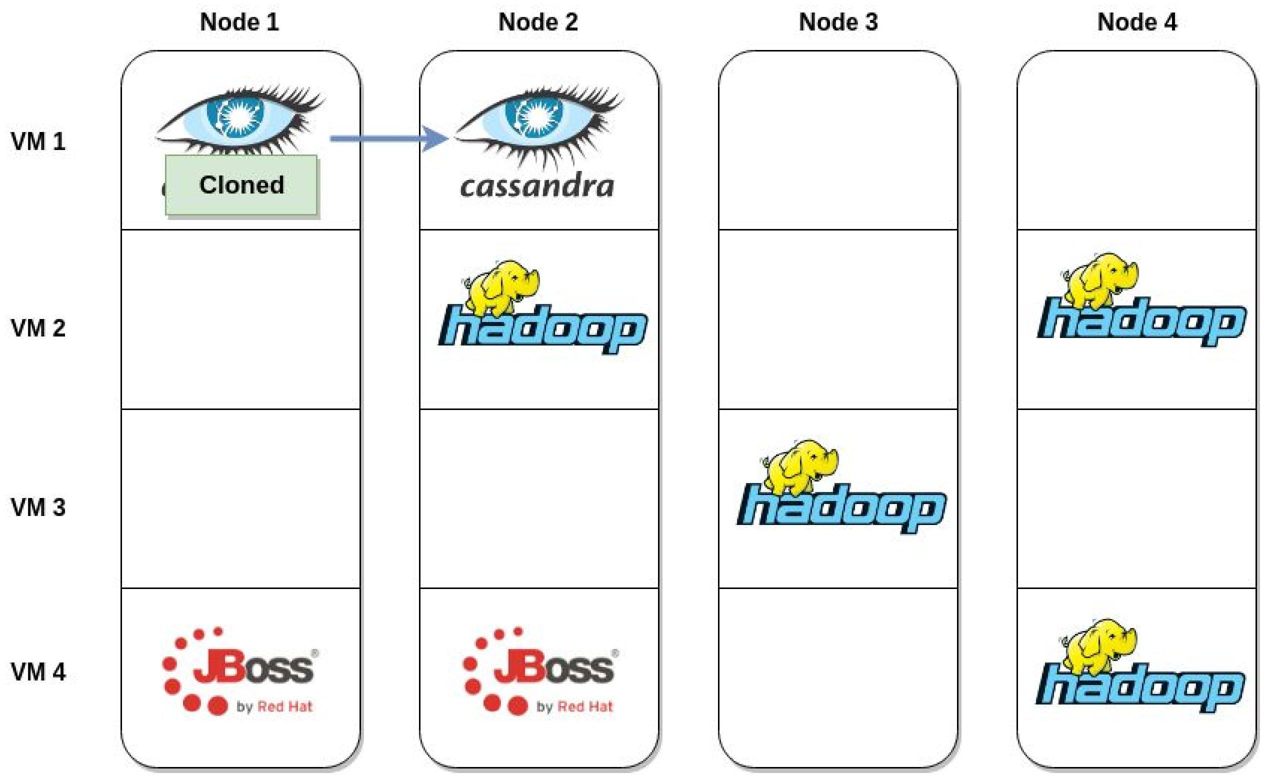
JVM Live Migration (real scenario)



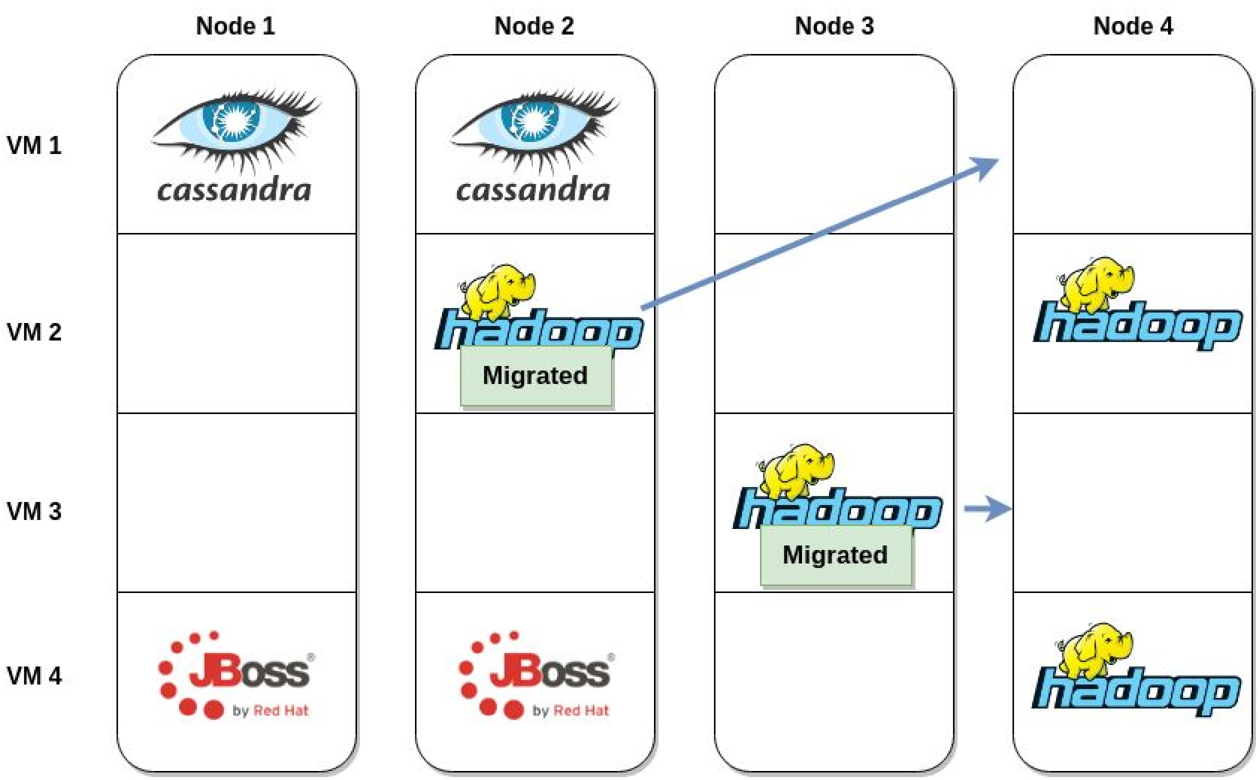
JVM Live Migration (real scenario)



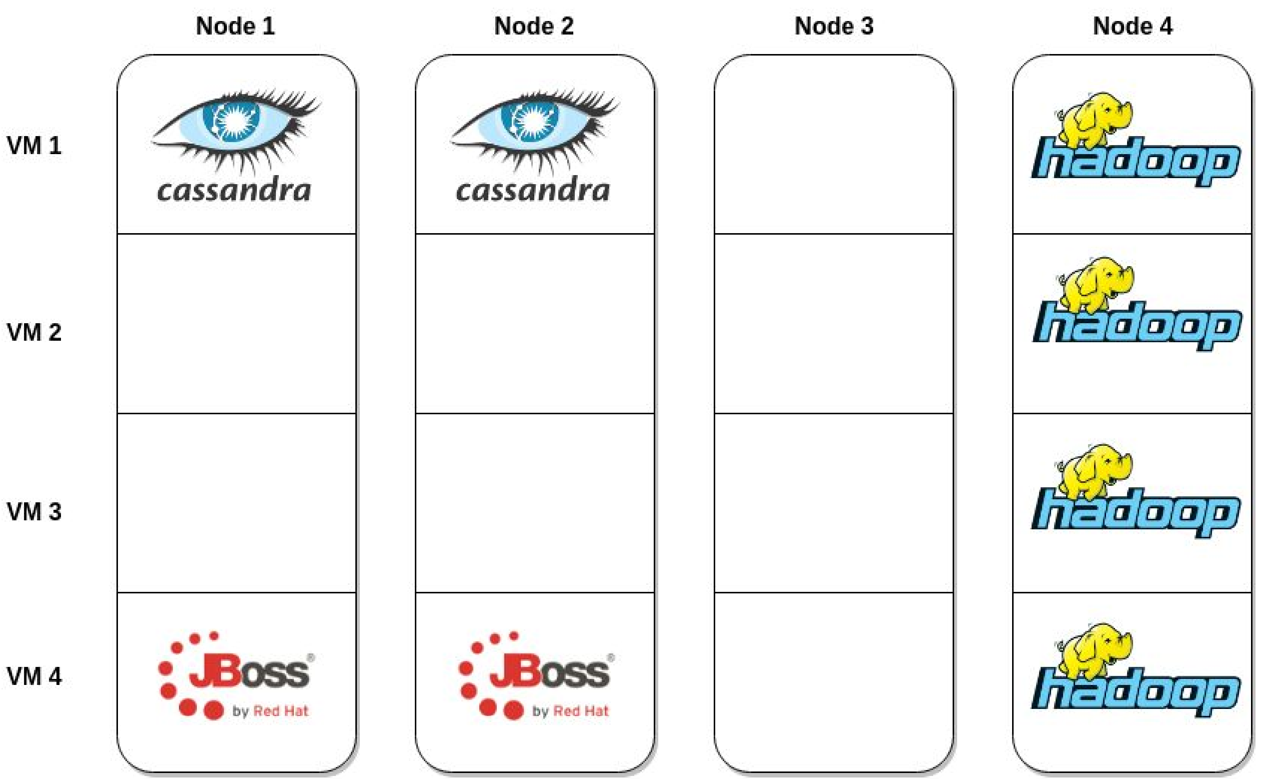
JVM Live Migration (real scenario)



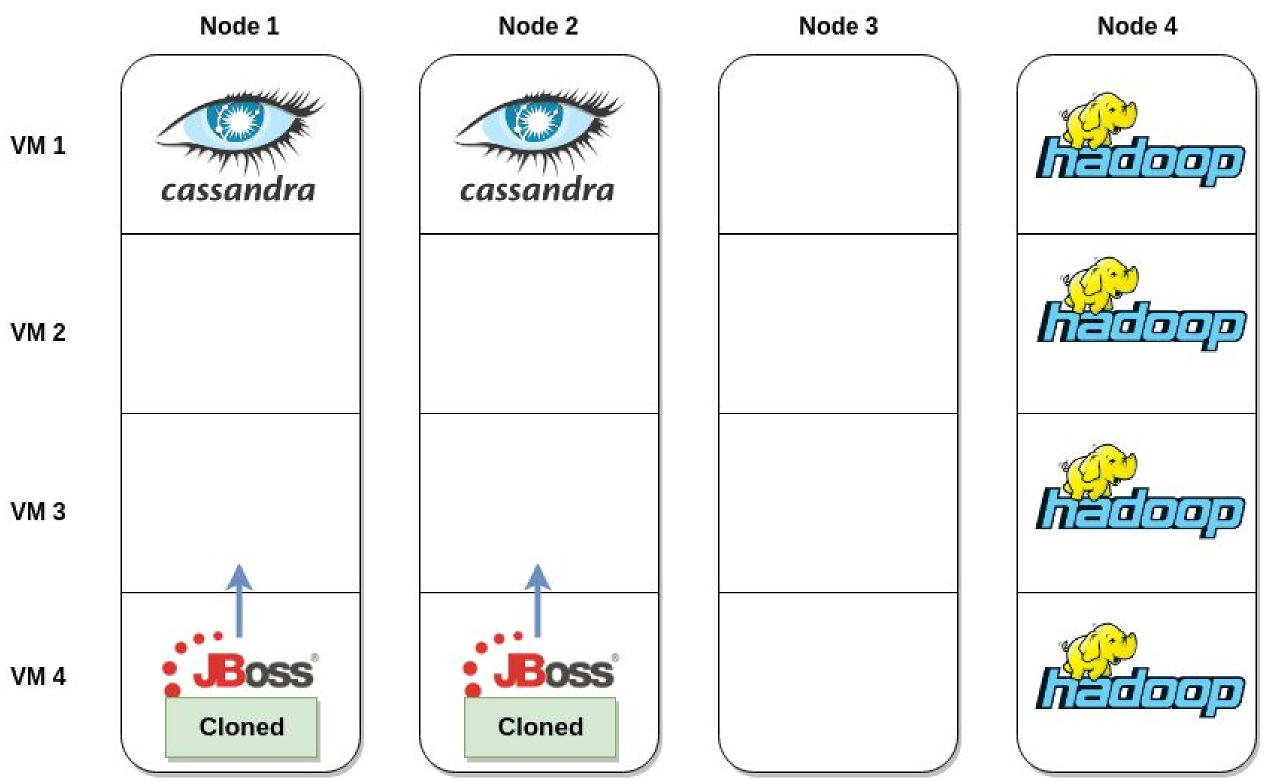
JVM Live Migration (real scenario)



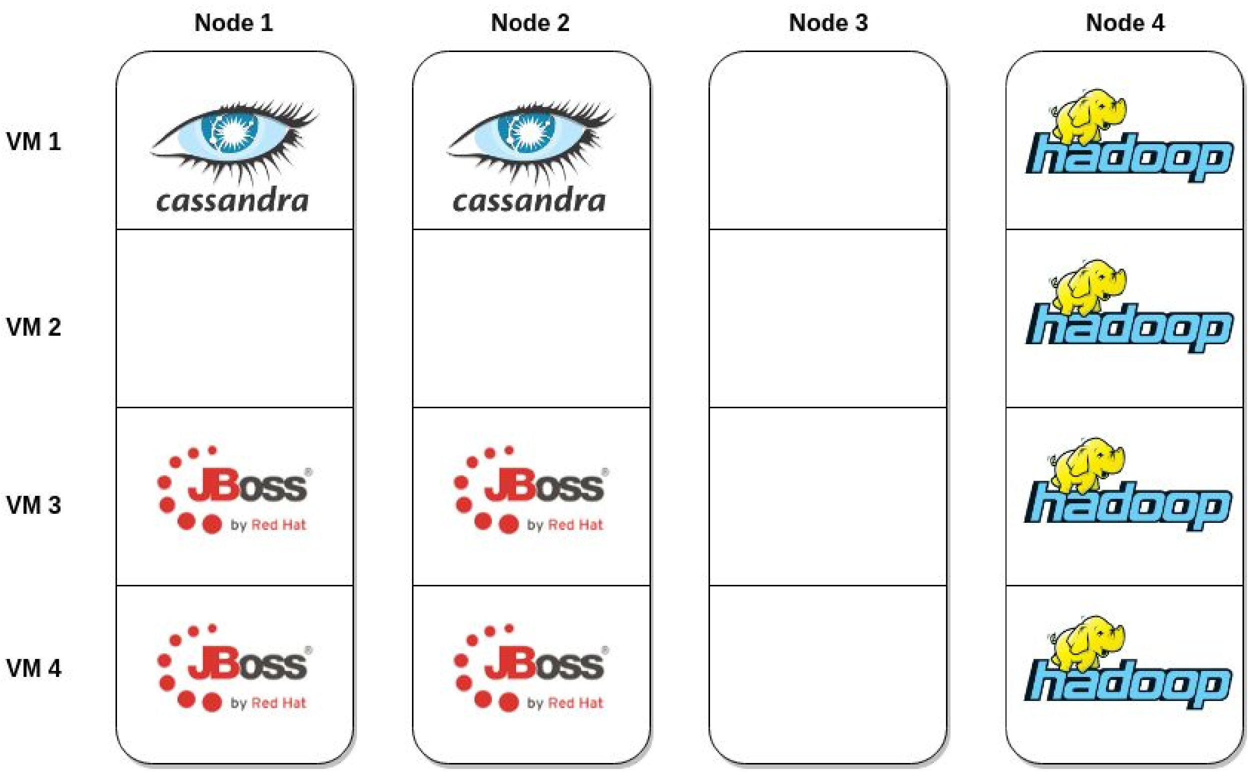
JVM Live Migration (real scenario)



JVM Live Migration (real scenario)



JVM Live Migration (real scenario)



Goals

Support JVM live migration with:

- ✓ Low total migration time;
- ✓ Low application downtime;
- ✓ Low application throughput impact;
- ✓ Low resource overhead;
- ✓ No programmer intervention;
- ✓ No special hardware/OS.

JVM Live Migration (challenges)

- Keep migration and application down times short;
- Avoid high resource (eg. CPU, Network) overhead;
- Avoid application slowdown / performance overhead;
- Cope with fast moving / allocation intensive applications;
- Cope with low/congested network bandwidths;

Drawbacks of Current Solutions

- ✗ Force application throttling (Clark et. al, 2005);
- ✗ Rely on high speed networks (Huang et. al, 2007);
- ✗ Fail to determine the live Working Set (Hou et. al, 2015);
- ✗ When only a process is targeted:
  - the whole system VM is migrated (containing multiple processes and kernel);
  - the whole process image is migrated (including unreachable data).
- ✗ Force full GC before migration (Kawachiya et. al, 2007);

ALMA - Key Insights

- Migrate only the process (JVM)
    - avoid kernel, other processes, etc;
- Use GC to reduce the snapshot size;
- Dynamically minimize the size of the memory to migrate
    - migrate only live objects
    - only collect regions which can be collected faster than transmitted through the network.

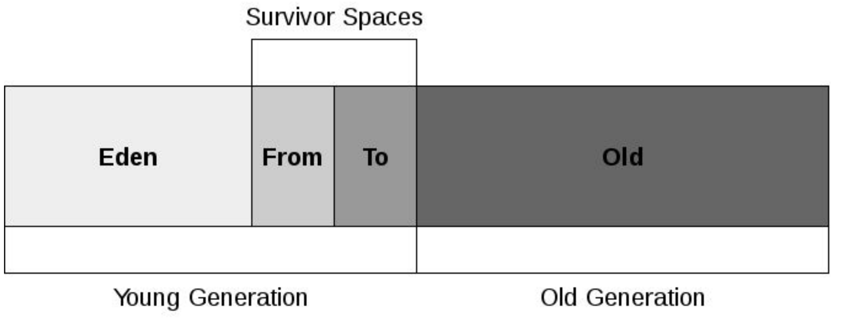
This leads to small (with almost only live data) snapshots.

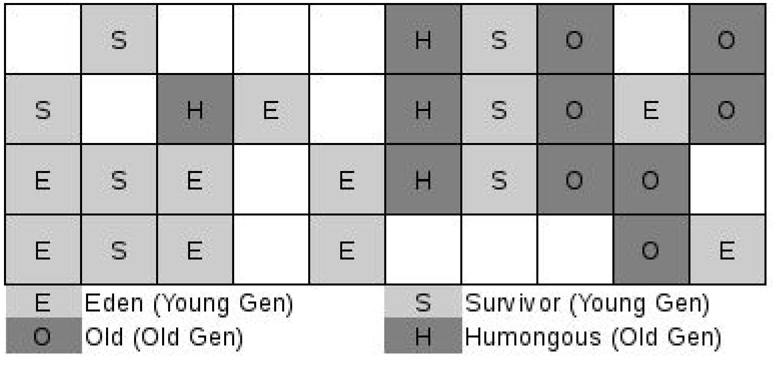
Presentation Overview

- GC background
- ALMA
  - Collection Set
  - Migration Workflow
  - Architecture
- Implementation
- Evaluation
  - App. Downtime
  - Total Migration Time
  - App. Throughput
  - Network Bandwidth Usage

GC Background

- **Parallel Scavenge** (old):
  - Spaces: Eden, Survivor, Old
  - Each space is a continuous memory block;
  - Young collection (only Eden and Survivor spaces), or
  - Full collection (all spaces)





- **G1** (most recent OpenJDK garbage collector):
  - Heap is divided into Regions (E,S,H,O)
  - Set of regions to collect: Collection Set (CS)

ALMA: Collection Set

Minimize size of snapshot

- Amount of data included in the snapshot:

$$Data = \sum_{Heap} used(r) - \sum_{CS} dead(r) \quad (1)$$

ALMA: Collection Set

Minimize size of snapshot

- Amount of data included in the snapshot:

$$Data = \sum_{Heap} used(r) - \sum_{CS} dead(r) \qquad (1)$$

- Total GCCost (time) for collecting the Collection Set (CS):

$$GCCost = \sum_{CS} cost(r) \qquad (2)$$

ALMA: Collection Set

Minimize size of snapshot

- Migration Cost (time) for migrating JVM:

$$MigrationCost = \frac{Data}{NetBandwidth} + GCCost \qquad (3)$$

ALMA: Collection Set

Minimize size of snapshot

- Migration Cost (time) for migrating JVM:

$$MigrationCost = \frac{Data}{NetBandwidth} + GCCost \qquad (3)$$

- GC Rate (amount of dead space collected per amount of time):

$$GCRate(r) = \frac{dead(r)}{cost(r)} \qquad (4)$$

ALMA: Collection Set

Minimize size of snapshot

- Migration Cost (time) for migrating JVM:

$$MigrationCost = \frac{Data}{NetBandwidth} + GCCost \qquad (3)$$

- GC Rate (amount of dead space collected per amount of time):

$$GCRate(r) = \frac{dead(r)}{cost(r)} \qquad (4)$$

- CS is the group of regions with GC Rate inferior to the Network Bandwidth:

$$CS = \{\forall r : GCRate(r) > NetBandwidth\} \qquad (5)$$

ALMA: Collection Set

Set of regions which can be collected faster than transmitted through the network:

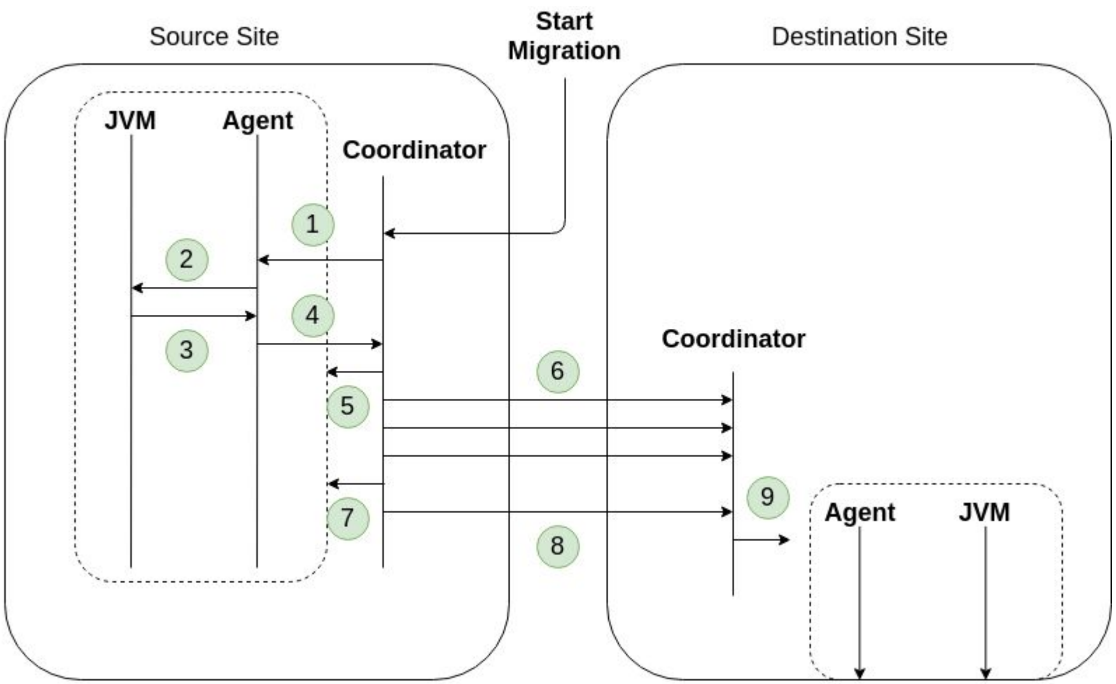
- Without collection, migration cost is X
- With collection, migration cost is X’ + GCCost

X > X’ + GCCost

- CS is the group of regions with GC Rate inferior to the Network Bandwidth:

$$CS = \{\forall r : GCRate(r) > NetBandwidth\} \qquad (5)$$

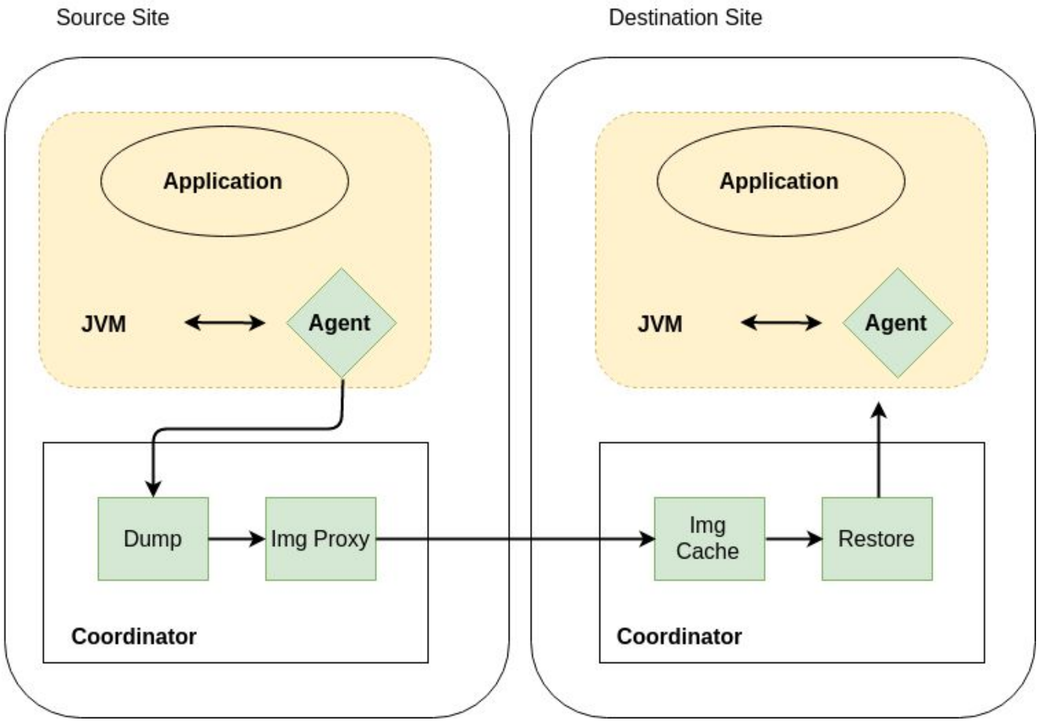
ALMA: Migration Workflow



Steps:

1. Prepare Snapshot
2. Build and Collect CS (Migr. Aware GC)
3. Return Free Mappings
4. Send Free Mappings to Coordinator
5. Checkpoint JVM
6. Send Snapshot
7. Stop JVM, incremental snapshot
8. Send final snapshot
9. Restore JVM from snapshot.

ALMA: Architecture



Components:

- **Application**: target application to migrate;
- **Agent**: analyzes the JVM;
- **Coordinator**: coordinates migration;
- **Dump**: takes JVM snapshots;
- **Img Proxy**: sends snapshot;
- **Img Cache**: caches snapshot;
- **Restore**: restores JVM from snapshots;

Implementation

- ALMA augmented G1 to support Migration Aware GC;
- Coordinator is implemented by extending CRIU to support remote migration. ALMA added two new components to CRIU:
  - Image Proxy - sends snapshot to the destination site;
  - Image Cache - caches snapshot in the destination site;
  - A patch is being iteratively refined to add both components to CRIU.

TÉCNICO LISBOA

gsd inesc id lisboa

Evaluation

- Evaluate ALMA’s performance compared to:
  - **CRIU** - Checkpoint and Restore for Linux;
  - **JAVMM** (Hou et. al, 2015) - Extends Xen to migrate Java applications. It simply collects the young generation before migration;
  - **ALMA-PS** - Similar to JAVMM but based on CRIU.
- Environment:
  - OpenStack VMs with 4vCPUs and 4GB RAM
  - DaCapo and SpecJVM2008 benchmark suites

Evaluation

Our Baseline

- performance compared to:
  - **CRIU** - Checkpoint and Restore for Linux
  - **JAVMM** (Hou et. al, 2015) - Extends Xen to migrate Java applications. It simply collects the young generation before migration.
  - **ALMA-PS** - Similar to JAVMM but based on CRIU;
- Environment:
  - OpenStack VMs with 4vCPUs and 4GB RAM
  - DaCapo and SpecJVM2008 benchmark suites

Evaluation

Our Baseline

- ... performance compared to:
    - **CRIU** - Check...

Targets JVM migration; Uses PS to reduce snapshot size

    - **JAVMM** (Hou et. al, 2015) - Extends Xen to migrate Java applications. It simply collects the young generation before migration.
    - **ALMA-PS** - Similar to JAVMM but based on CRIU;
- Environment:
    - OpenStack VMs with 4vCPUs and 4GB RAM
    - DaCapo and SpecJVM2008 benchmark suites

Evaluation

Our Baseline

- performance compared to:
  - **CRIU** - Check

Targets JVM migration; Uses PS to reduce snapshot size

  - **JAVMM** (Hou et. al, 2015) - Extends Xen to migrate Java
    n before
    migratio

Similar to ALMA, but using PS (as in JVMM)

  - **ALMA-PS** - Similar to JAVMM but based on CRIU;
- Environment:
  - OpenStack VMs with 4vCPUs and 4GB RAM
  - DaCapo and SpecJVM2008 benchmark suites

Evaluation

- Application Downtime;
- Total Migration Time;
- Application Throughput;
- Network Bandwidth Usage;
- Migration-aware GC vs G1 GC (refer to paper)
- ALMA with more resources (refer to paper)

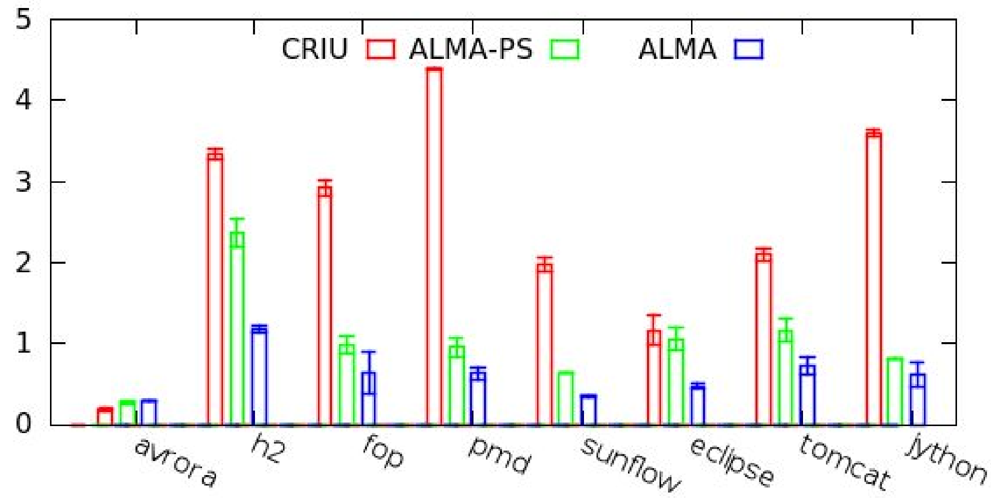
Evaluation

- Application Downtime;
- Total Migration Time;
- Application Throughput;
- Network Bandwidth Usage;
- Migration-aware GC vs G1 GC (refer to paper)
- ALMA with more resources (refer to paper)

These metrics measure the impact of migration on application performance.

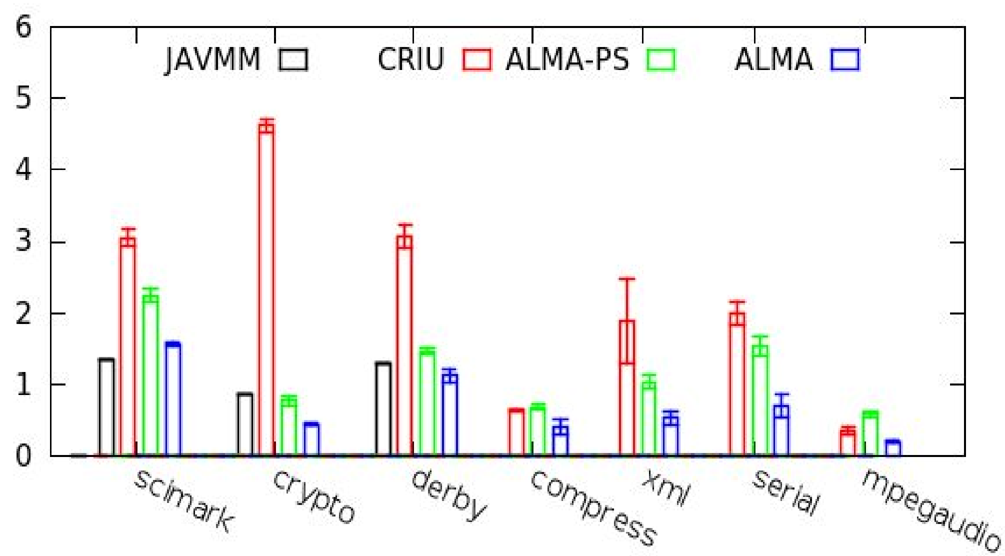
Evaluation - Application Downtime (seconds)

DaCapo



The Smaller the Better!

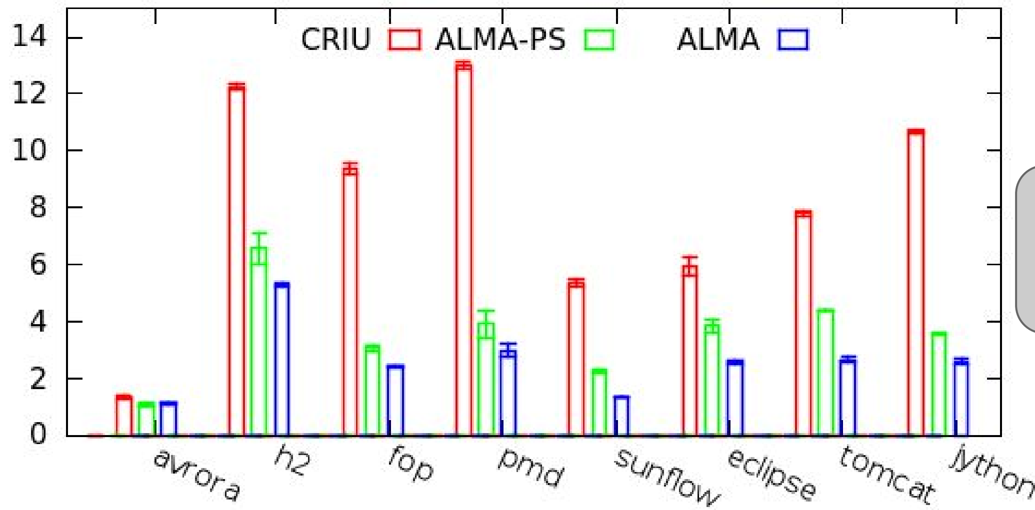
SPECjvm2008



The Smaller the Better!

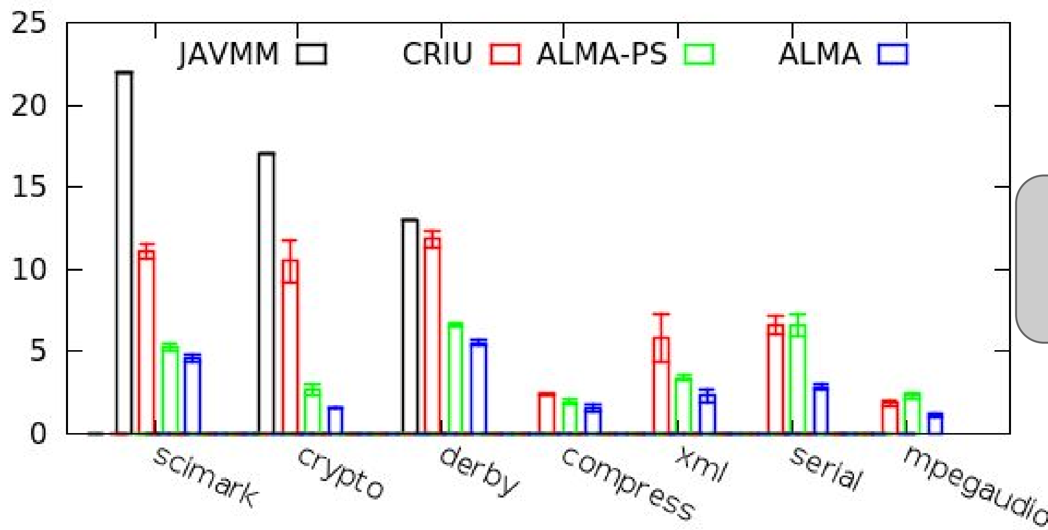
Evaluation - Total Migration Time (seconds)

DaCapo



The Smaller the Better!

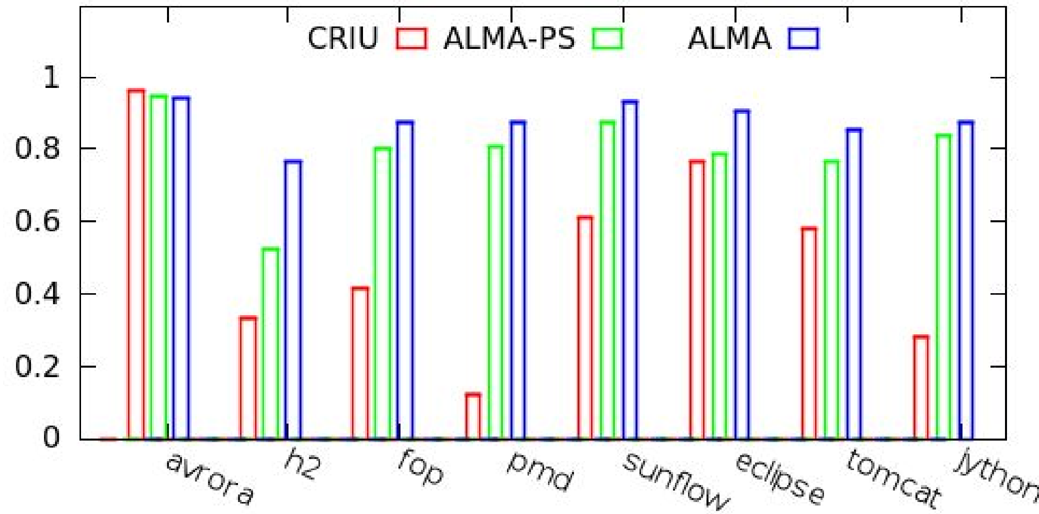
SPECjvm2008



The Smaller the Better!

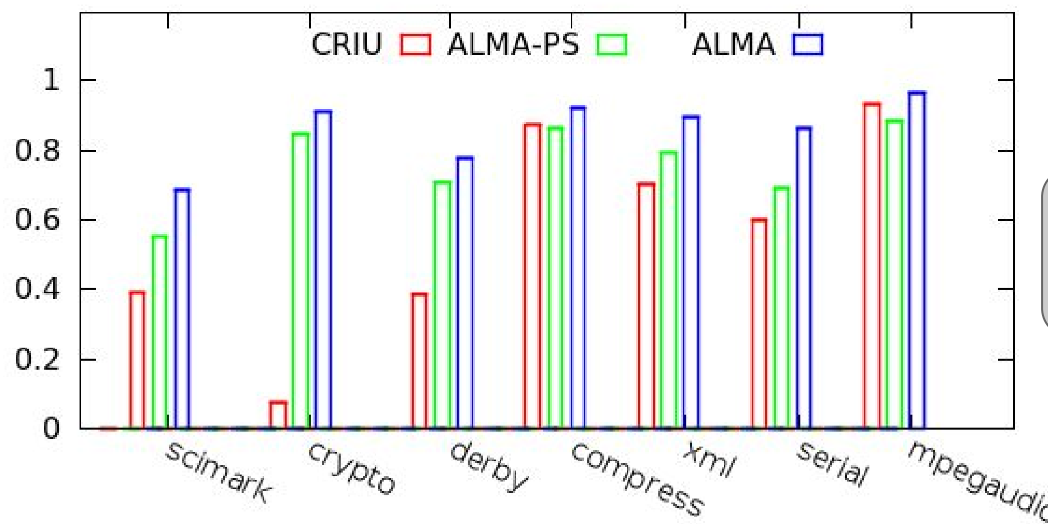
Evaluation - Application Throughput (normalized)

DaCapo



The Higher the Better!

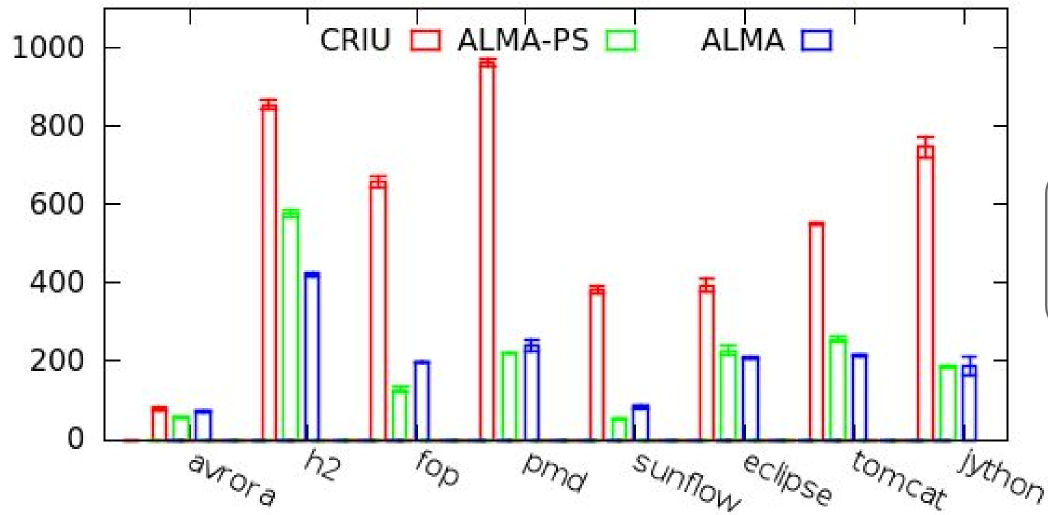
SPECjvm2008



The Higher the Better!

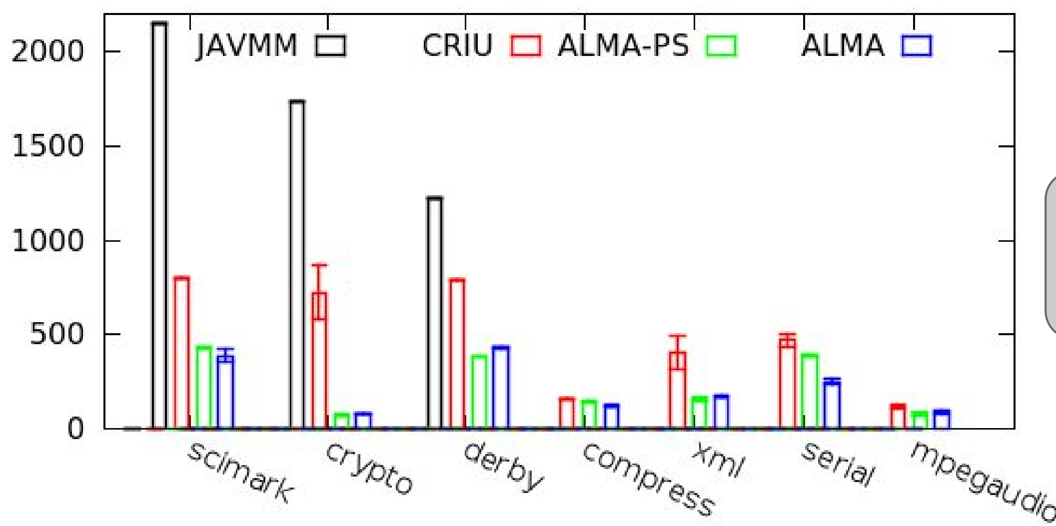
Evaluation - Network Bandwidth Usage (MBs)

DaCapo



The Smaller the Better!

SPECjvm2008



The Smaller the Better!

Conclusions

- ALMA offers efficient migration of Java server applications
  - by selectively avoiding garbage when it pays off
- ALMA's implementation is based on OpenJDK and CRIU;
  - Code is available at: github.com/rodrigo-bruno/alma
- ALMA outperforms current solutions in:
  - Reducing application overhead
  - Reducing total migration time and downtime
  - Reducing network bandwidth usage

Thank you for your time.
Questions?

Rodrigo Bruno
email: rodrigo.bruno@tecnico.ulisboa.pt
webpage: www.gsd.inesc-id.pt/~rbruno
alma's github: github.com/rodrigo-bruno/alma