MEDEA: Towards Expressive Scheduling of Long-Running Applications

Panagiotis Garefalakis[†◇] Konstantinos Karanasos[†] Peter Pietzuch[◇]
Arun Suresh[†] Sriram Rao[†]

[†]*Microsoft* [◇]*Imperial College London*

Applications with long-running containers are increasingly common in shared production clusters. Typical examples include streaming, machine learning, and latency-sensitive applications with long-running executors for their tasks, as well as traditional online services. In fact, within Microsoft, we witness a substantial shift towards *long-running applications (LRAs)* in production clusters: analyzing four of Microsoft's big data clusters we found that the fraction of LRAs was ranging from 9% to above 30%.

These applications have stringent and complex scheduling requirements: application owners want placement decisions to respect complex constraints, e.g., co-locating or separating long-running containers across node groups, accounting for *performance* and *resilience*; cluster operators must also achieve global objectives, such as minimizing *future regret*, i.e., avoiding decisions that hinder placement of upcoming applications. All of this must be done without affecting the scheduling latency of short-running containers.

Despite the evolution from predominantly batch jobs to a heterogeneous mix of applications, production clusters still optimize for achieving low-latency for batch jobs, ignoring the specific requirements of LRAs.

To this end, we propose MEDEA, a new cluster scheduler with dedicated support for LRAs. MEDEA follows a two-scheduler design: (i) for LRAs, it applies an optimization-based approach with expressive placement constraints that captures interactions between containers within and across applications, while minimizing future regret; (ii) for short-running containers, MEDEA uses a traditional task-based scheduler to achieve low placement latency. Our implementation as part of Apache Hadoop/YARN shows that MEDEA places LRAs effectively and in a scalable manner.

EuroSys '17, April 23–26, 2017, Belgrade, Serbia.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

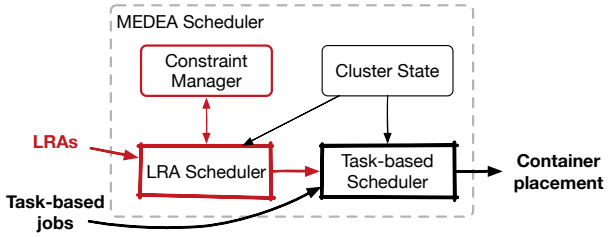


Figure 1: MEDEA Scheduler design

Key features MEDEA addresses the aforementioned requirements based on two novel features:

(i) expressive, high-level constraints. We introduce the notions of container tags and node groups to allow applications to concisely express powerful constraints within and across applications. Constraints can refer to both existing and future applications, and no knowledge of the underlying cluster's organization is required.

(ii) two-scheduler design. We formulate the scheduling of LRAs as an optimization-based problem. As shown in Figure 1, MEDEA uses a dedicated scheduler for the LRA placement, whereas task-based applications are passed directly to a traditional scheduler. This design allows us to perform careful placement of LRAs, which can tolerate relatively higher scheduling latencies, without impacting the scheduling of task-based applications.

Implementation and evaluation We implemented MEDEA as an extension to Apache Hadoop/YARN, one of the most widely used production cluster managers. We show that on a 274-node cluster, MEDEA, thanks to its expressive constraints, can reduce tail latency for HBase requests by 3× and increase its throughput by 34% compared to YARN. Coupling MEDEA with resource isolation mechanisms, such as cgroups, can further improve its performance. Unlike single-scheduler designs, we show that MEDEA does not impact the scheduling latency of task-based applications. We are currently in the process of contributing MEDEA components to Apache Hadoop YARN[1].

[1] https://issues.apache.org/jira/browse/YARN-5468

Expressive Scheduling of Long-Running Applications

Panagiotis Garefalakis[1,2], Konstantinos Karanasos[1], Peter Pietzuch[2], Arun Suresh[1], Sriram Rao[1]

[1]Microsoft [2]Imperial College London

MOTIVATION

Long-running applications (LRAs) = applications with long-running containers, running for hours, days or even months

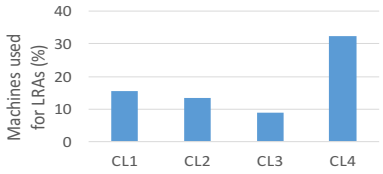
LRAs are increasingly common:

- Interactive data-intensive applications, e.g., Spark, Impala, Hive
- Streaming systems, e.g., Storm, Heron, Flink, Millwheel
- Latency-sensitive applications, e.g., HBase, Memcached
- ML frameworks, e.g. Spark ML-Lib and SystemML

GOAL

Place a given LRA in the cluster in a way that:

1. Meets the complex LRA requirements
2. Optimizes for future regret
3. Does not affect scheduling of traditional task-based apps

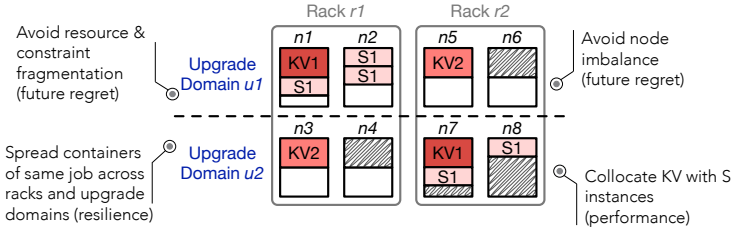


Machines used for LRAs in four Microsoft clusters

SCHEDULING REQUIREMENTS

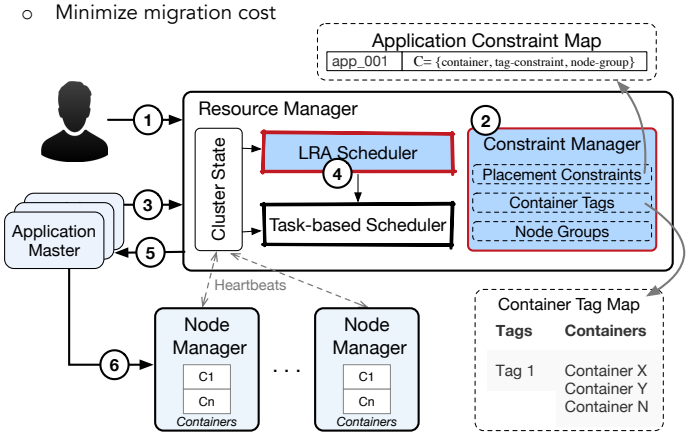
1. Application owners
   Improve LRA performance and resilience
2. Cluster operators
   Place LRAs in a way that does not hinder placement of future applications (minimize future regret)
3. Scheduling latency
   LRAs can tolerate higher scheduling latencies, but short-running containers scheduling should not be affected

Key/value store instances and streaming job example



LRA SCHEDULING

- Optimization-based scheduling, formed as an ILP problem
- Support for both *soft* and *hard* constraints
- Objective function: minimize future regret
  - Node imbalance, resource & constraint fragmentation
- Re-planning of LRAs
  - Minimize migration cost



PLACEMENT CONSTRAINTS

Expressive

Intra- and inter-application constraints
Affinity/anti-affinity/cardinality constraints

High-level

Refer to current and future LRAs
Do not require knowledge of the cluster's organization

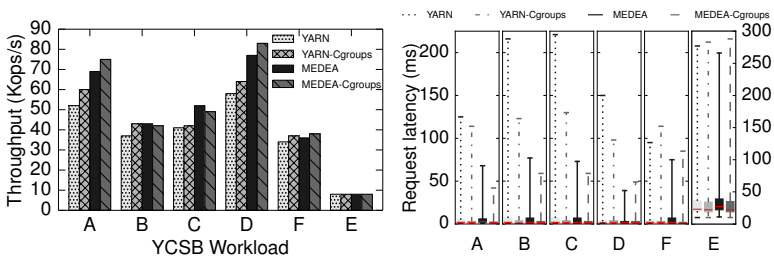
Constraint expression:
C = [container, tag-constraint, node-group],
where tag-constraint = { tag, c_{min}, c_{max} }

- Affinity: C_{af} = [tID_002, { appID_0023 **AND** JM, 1, ∞ }, node]
- Anti-affinity: C_{aa} = [tID_002, { heron, 0, 0 }, upgrade_domain]
- Group cardinality: C_{gc} = [*, { IO_critical, 0, 10 }, rack]

Nodes/node groups	Example tags
node n1	memcached, appID_0023, heron, appID_0037
rack r1	memcached, appID_0023, appID_0024, heron

Node group	Example cluster node sets
node	{n1}, {n2}, {n3}, {n4}, {n5}, {n6}, {n7}, {n8}
rack	{n1, n2, n3, n4}, {n5, n6, n7, n8}
upgrade domain	{n1, n2, n5, n6}, {n3, n4, n7, n8}

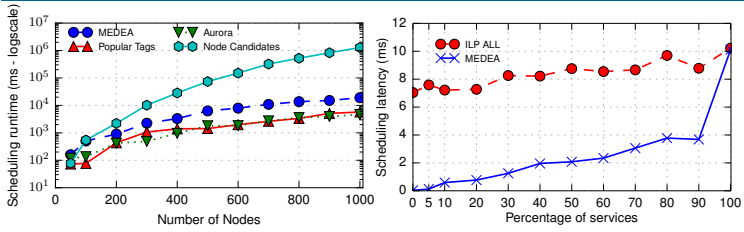
Performance benefit of application constraints



HBase stressed with YCSB, MR jobs generated with GridMix on a 274 machine cluster

- *MEDEA* achieves up to **53%** better **throughput** (35% to YARN+cgroups)
- *MEDEA* achieves up to **3.9x** better **tail latency** (99th percentile)

Two-scheduler design benefit



- Scalability: comparable latency with greedy approaches (LRAs 20% of cluster)
- Two-scheduler design: **9.5x** better latency when LRAs are 20% of workload

The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) is gratefully acknowledged.