MEDEA

Scheduling of Long Running Applications in Shared Production Clusters

Panagiotis Garefalakis
Imperial College London
pg1712@imperial.ac.uk

Konstantinos Karanasos
Microsoft
kokarana@microsoft.com

Peter Pietzuch
Imperial College London
prp@imperial.ac.uk

Arun Suresh
Microsoft
arsuresh@microsoft.com

Sriram Rao
Microsoft
sriramra@microsoft.com

Long-Running Applications (LRAs)

Richer applications in compute clusters

> **Short-running containers**
- MapReduce, Scope, Tez

Shift towards long running containers

> **Interactive data-intensive** applications
- Spark, Hive LLAP

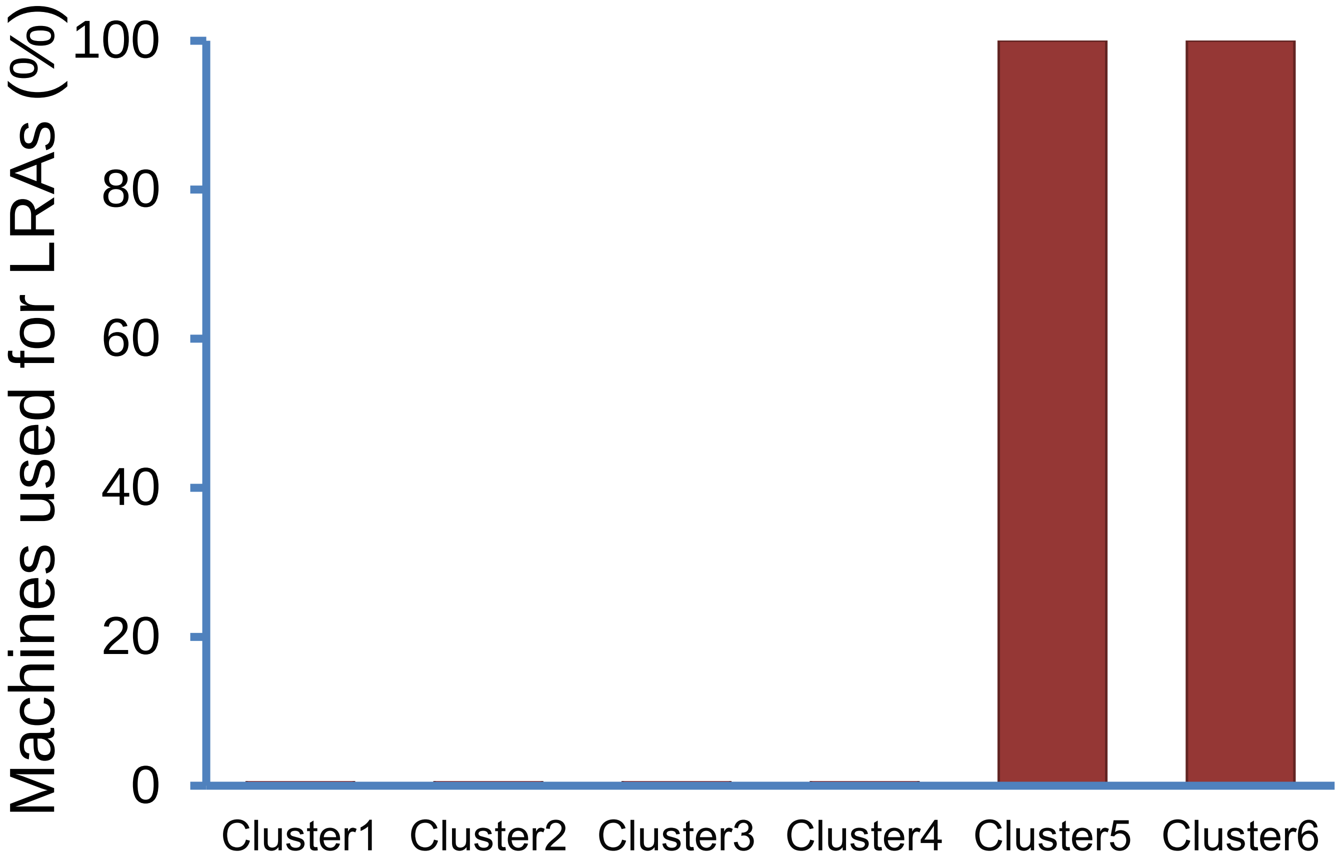
> **Streaming** systems
- Flink, Storm, SEEP

> **Latency-sensitive** applications
- HBase, Memcached

> **ML** frameworks
- TensorFlow, Spark ML-lib

LRAs = applications with long-running containers (running from hours to months)

LRAs in Microsoft’s analytics clusters



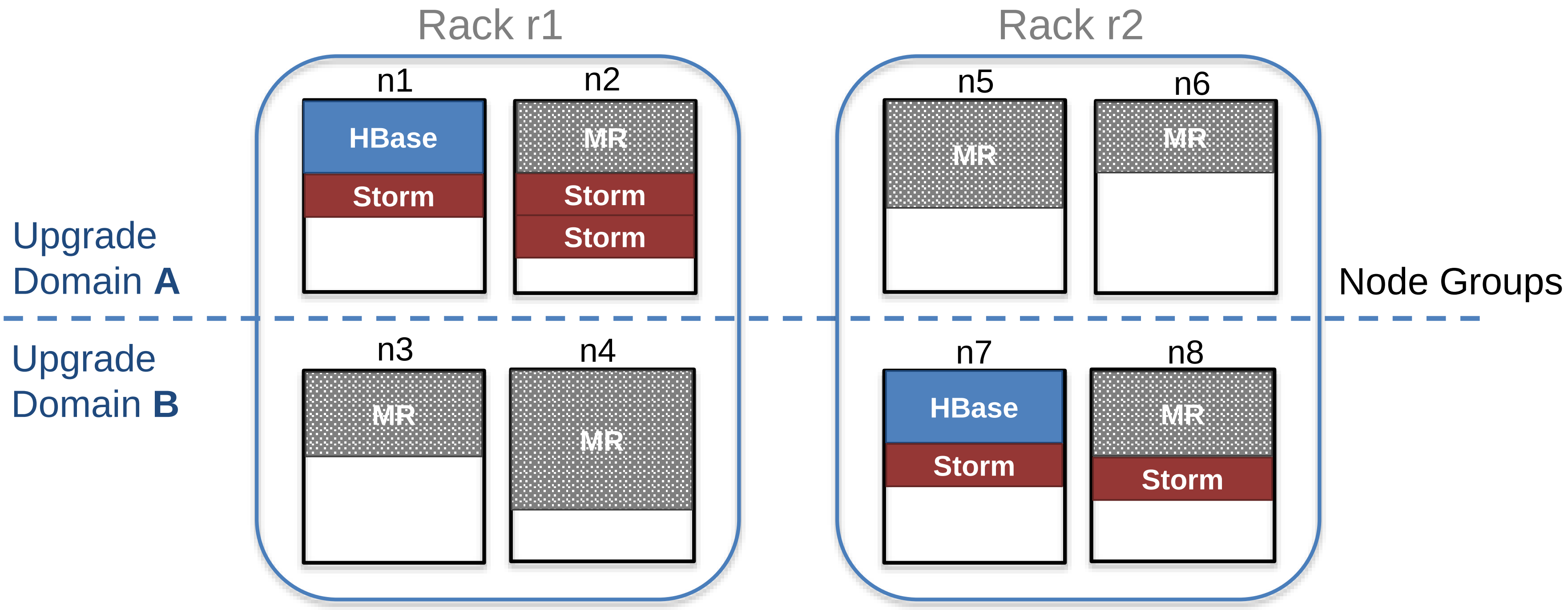
> Each cluster comprises tens of thousands of machines

> **10-100**% of each cluster’s machines used for LRAs

> Machines for LRAs are picked statically or randomly

LRA placement is important

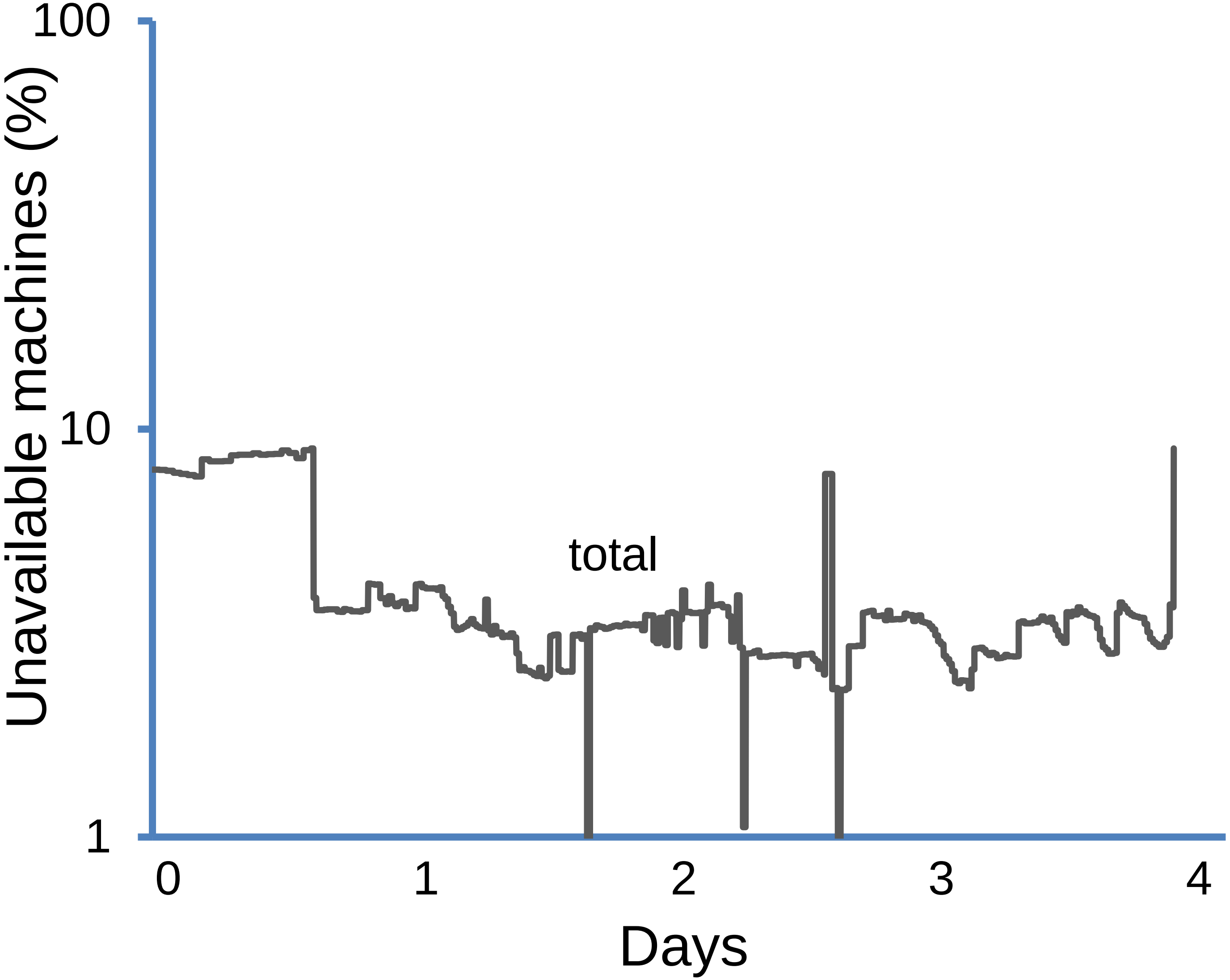
LRA scheduling problem



- **Performance:** “Place Storm containers in the same rack as HBase”
- **Cluster objectives:** “Minimize resource fragmentation”

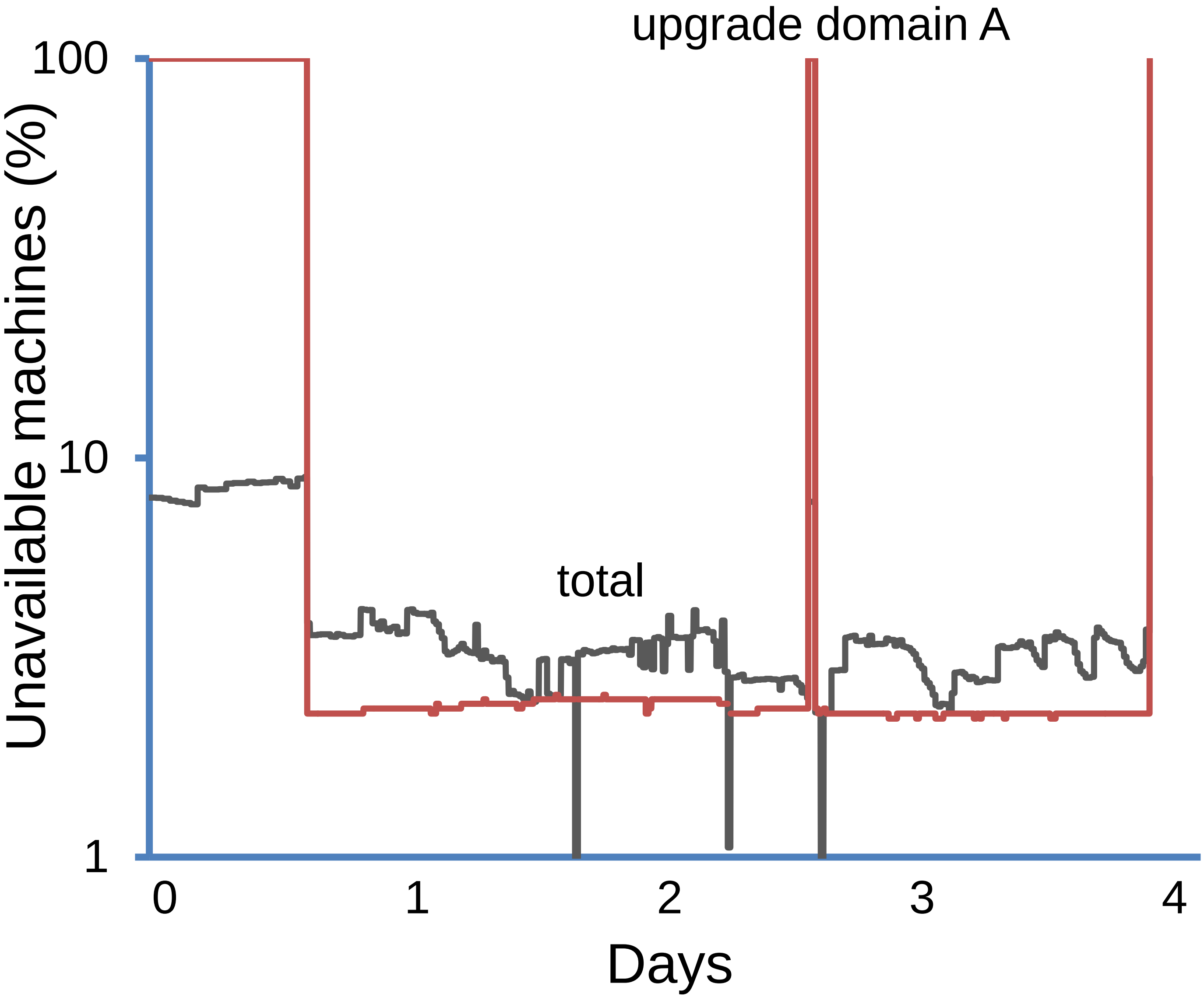
Goal: Diligent placement through constraints

Machine unavailability in a Microsoft cluster



- Cluster organized in node groups

Machine unavailability in a Microsoft cluster



- Cluster organized in node groups
- Machines become unavailable in groups
- With **random placement**, an LRA might lose all containers at once

Challenges

- How to relate containers to node groups?

- How to express different types of constraints related to LRA containers?

- How to achieve high quality placement without affecting task-based jobs?

MEDEA

LRA scheduling with expressive placement constraints

> How to relate containers to node groups?

⚖ Support container tags and logical node groups

> How to express different types of constraints related to LRA containers?

⚖ Introduce expressive cardinality constraints

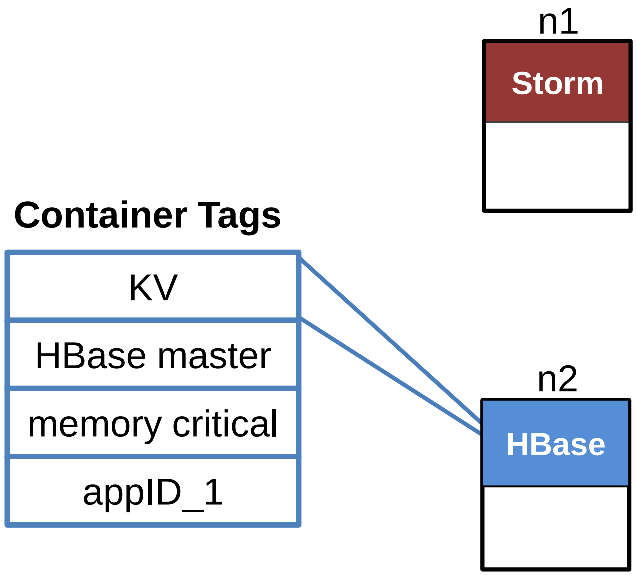
> How to achieve high quality placement without affecting task-based jobs?

⚖ Follow a two-scheduler design

Container tagging

> **Idea:** use container tags to refer to group of containers

- Describe
  - application type
  - application role
  - resource specification
  - global application ID

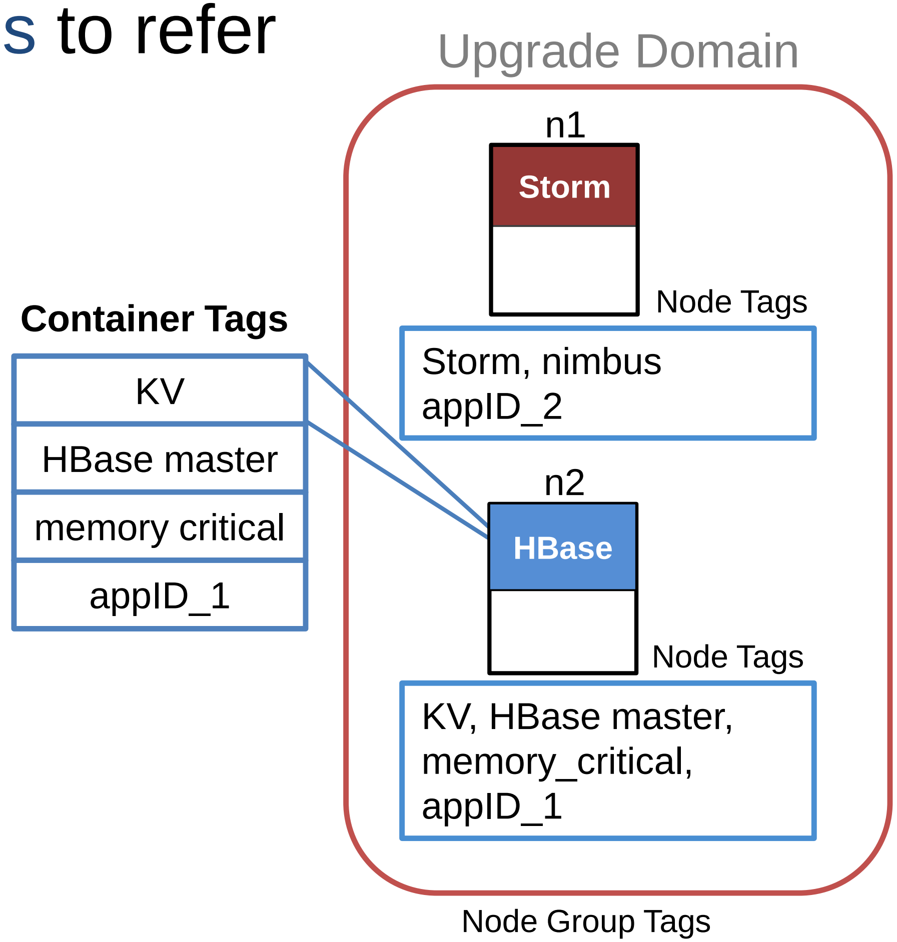


- Can refer to any current or future LRA container

Hierarchical grouping of nodes

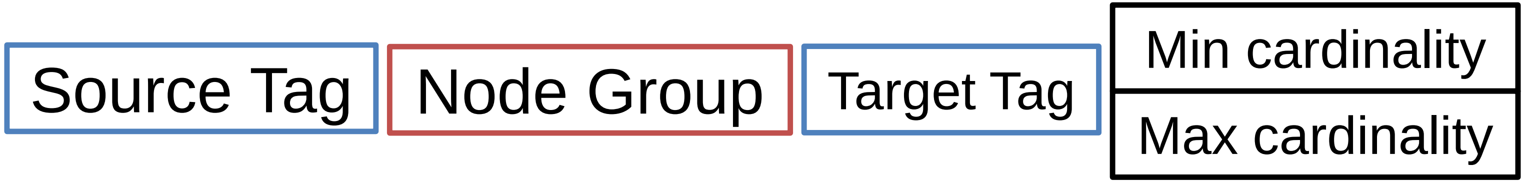
> **Idea:** logical node groups to refer to dynamic node sets

- E.g. node, rack, upgrade domain
- Associate nodes with all the container tags that live there
- Hide infrastructure “spread across upgrade domains”



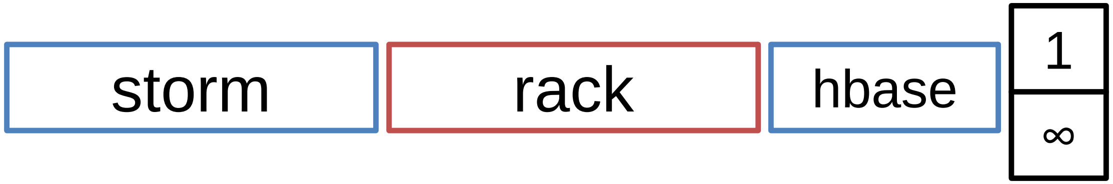
Defining constraints

> Generic constraints to capture a variety of cases

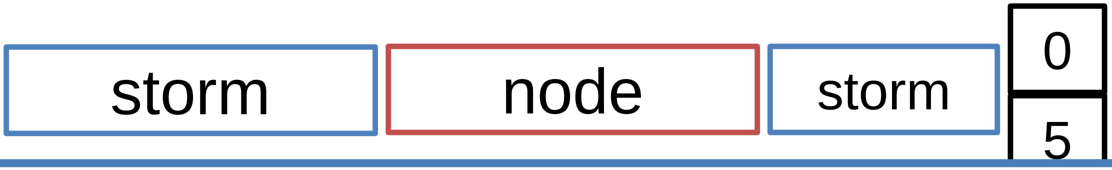

Source Tag
Node Group
Target Tag
Min cardinality
Max cardinality

Min cardinality ≤ occurrences (Target Tag) ≤ Max cardinality

> **Affinity** “Place Storm containers in the same rack as HBase”


storm
rack
hbase
1
∞

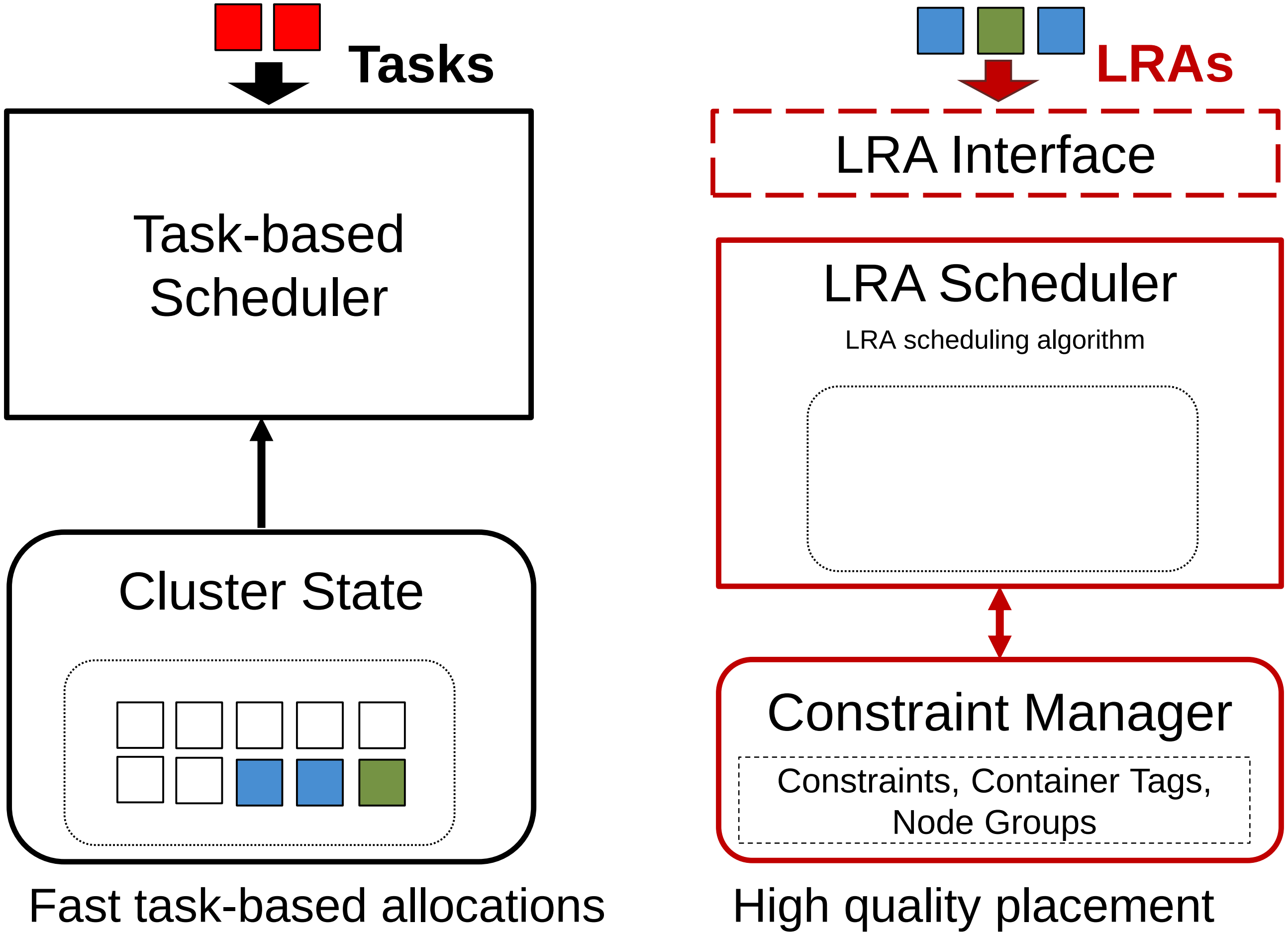
> **Cardinality** “Place up to 5 Storm containers in the same node”


storm
node
storm
0
5

A single constraint type is sufficient!

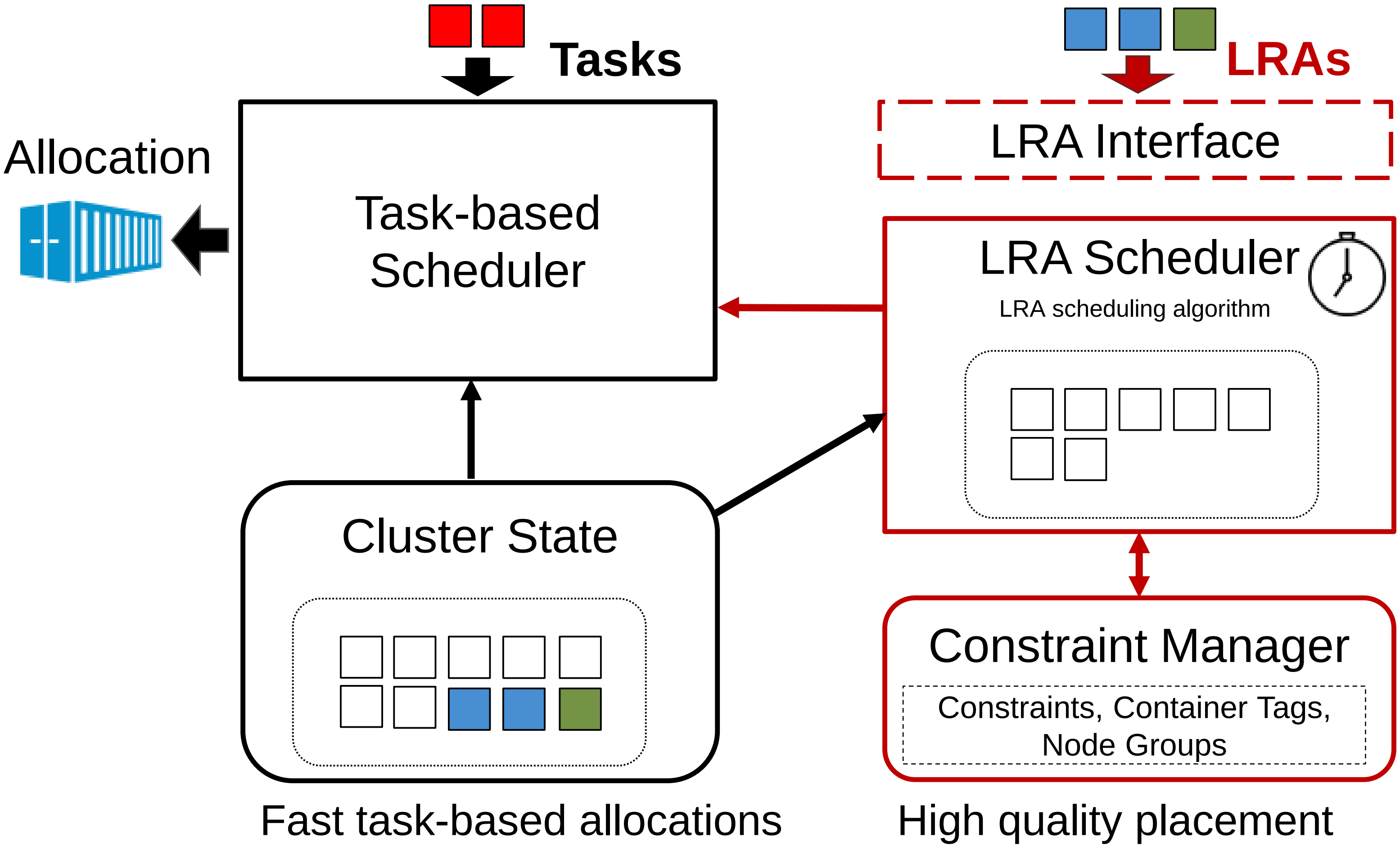
Two-scheduler design

> **Idea**: traditional scheduler for task-based jobs, optimization-based scheduler for LRAs



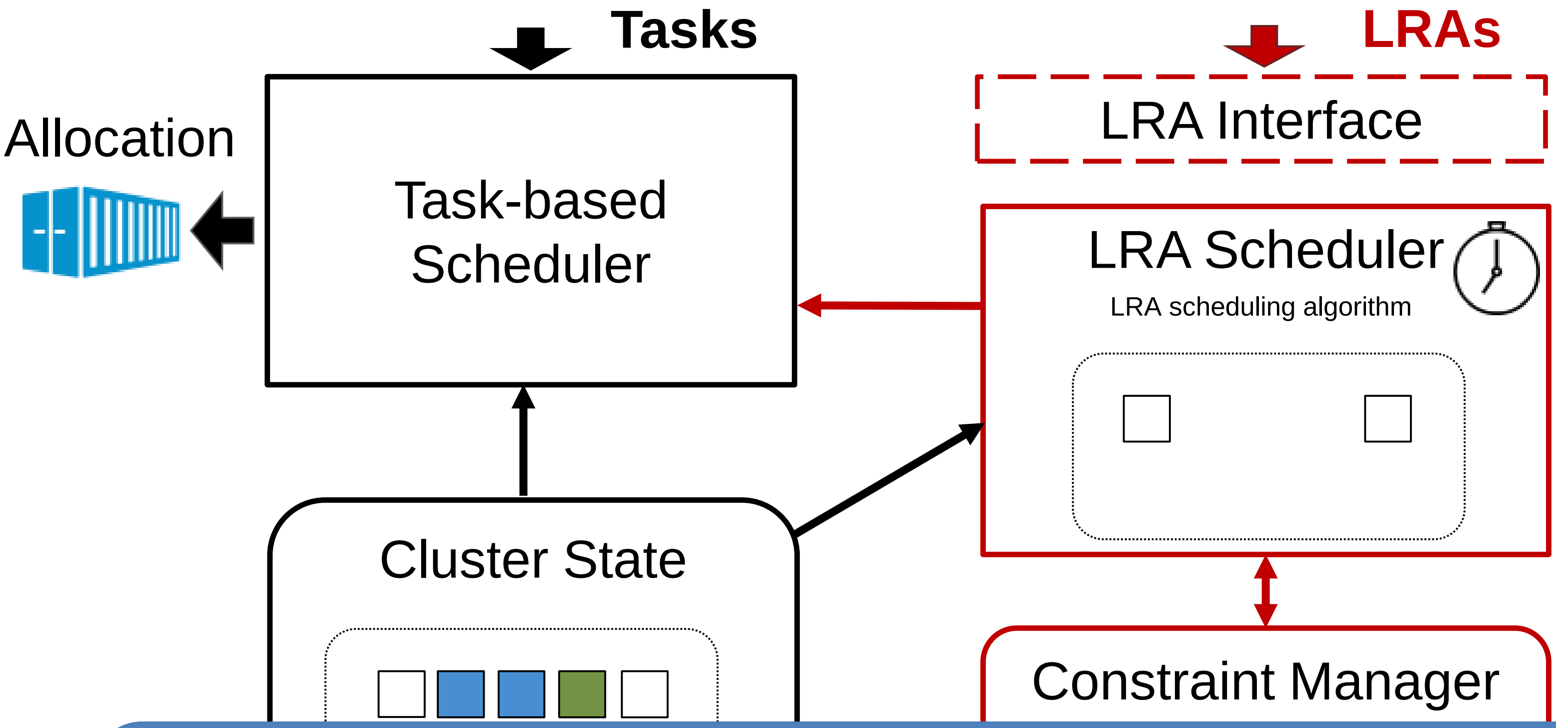
Two-scheduler design

> ILP scheduling algorithm with flexible optimization goals
    - Invoked at configurable intervals, considers multiple containers



Two-scheduler design

> ILP scheduling algorithm with flexible optimization goals
  - Invoked at configurable intervals, considers multiple containers



Satisfy LRA constraints without affecting task-based jobs

Implementation

- Built as an extension to APACHE hadoop **YARN** code is part of current release 3.1.0

- Introduced APIs for clients to specify constraints, MEDEA's components added to Resource Manager

- YARN's Capacity Scheduler was used as task-based scheduler

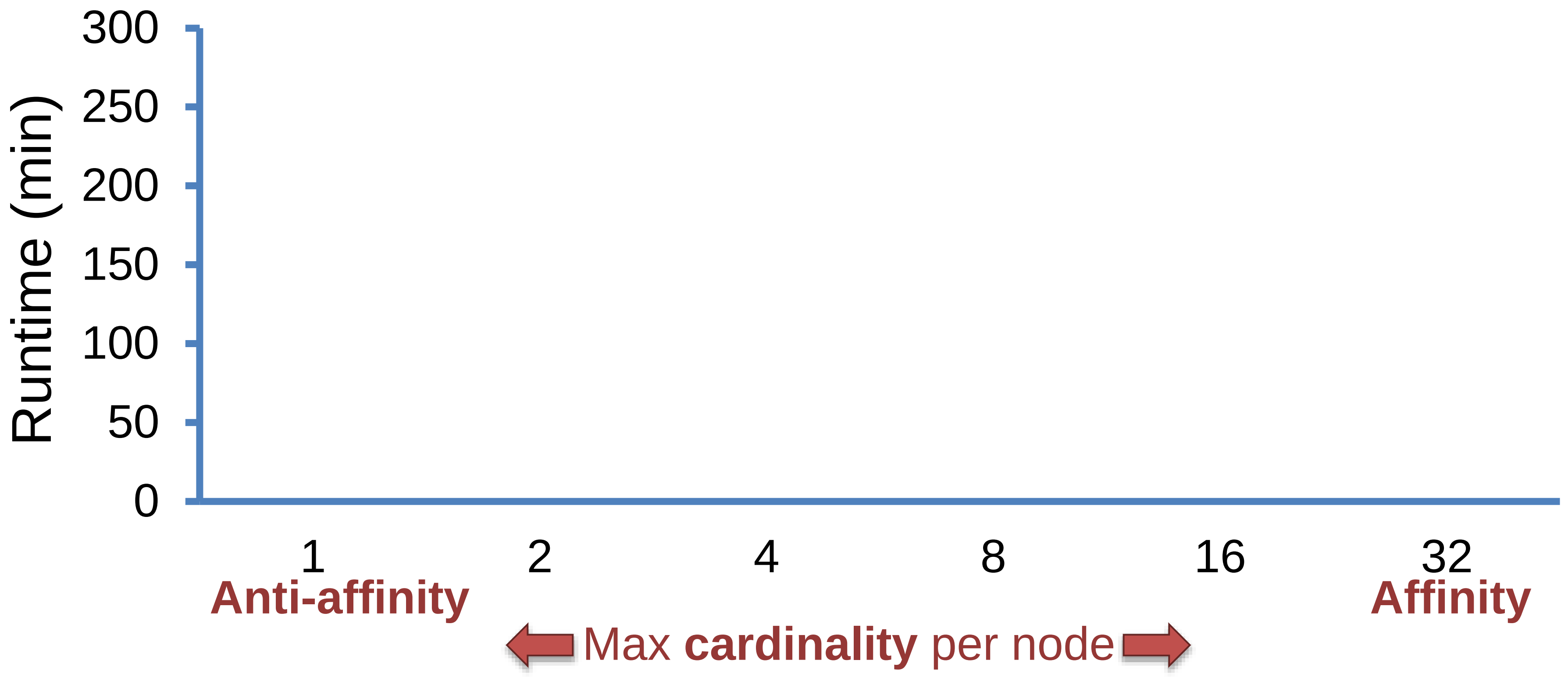
Evaluation

- What is the impact of cardinality constraints in LRA performance?

- What is the two-scheduler benefit in terms of scheduling latency?

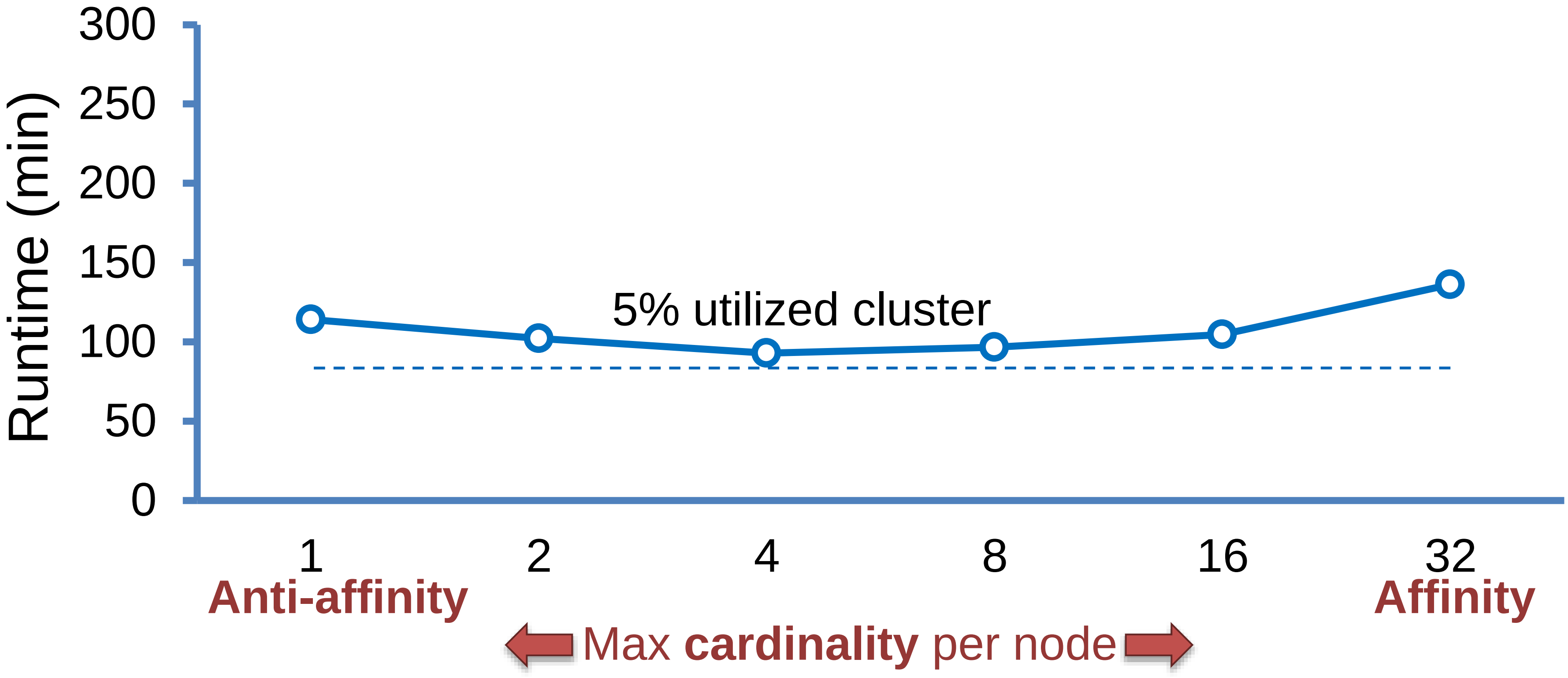
- What is the benefit of MEDEA in a large scale environment?

Impact of cardinality in LRA performance



> **TensorFlow** ML workflow with 1M iterations using 32 workers with varying workers per node

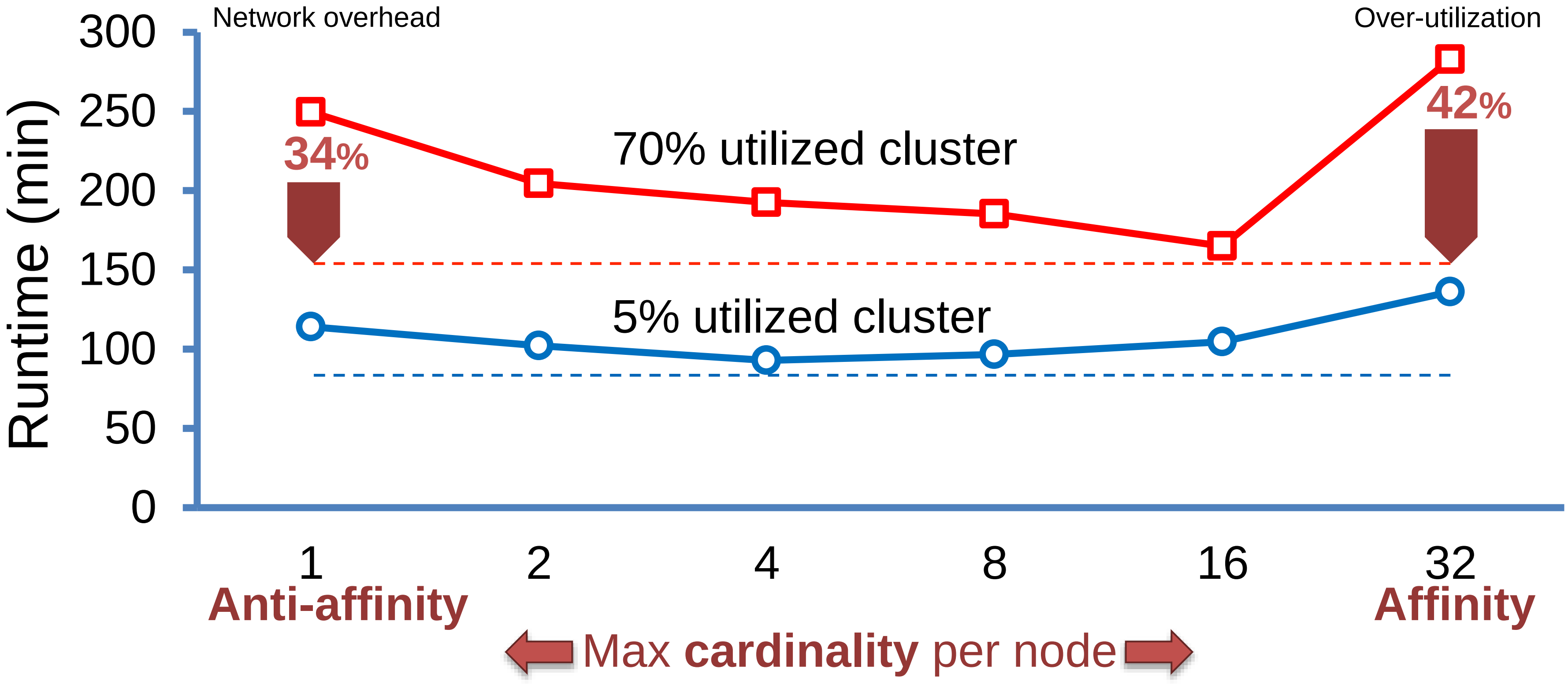
Impact of cardinality in LRA performance



Cardinality constraints are important

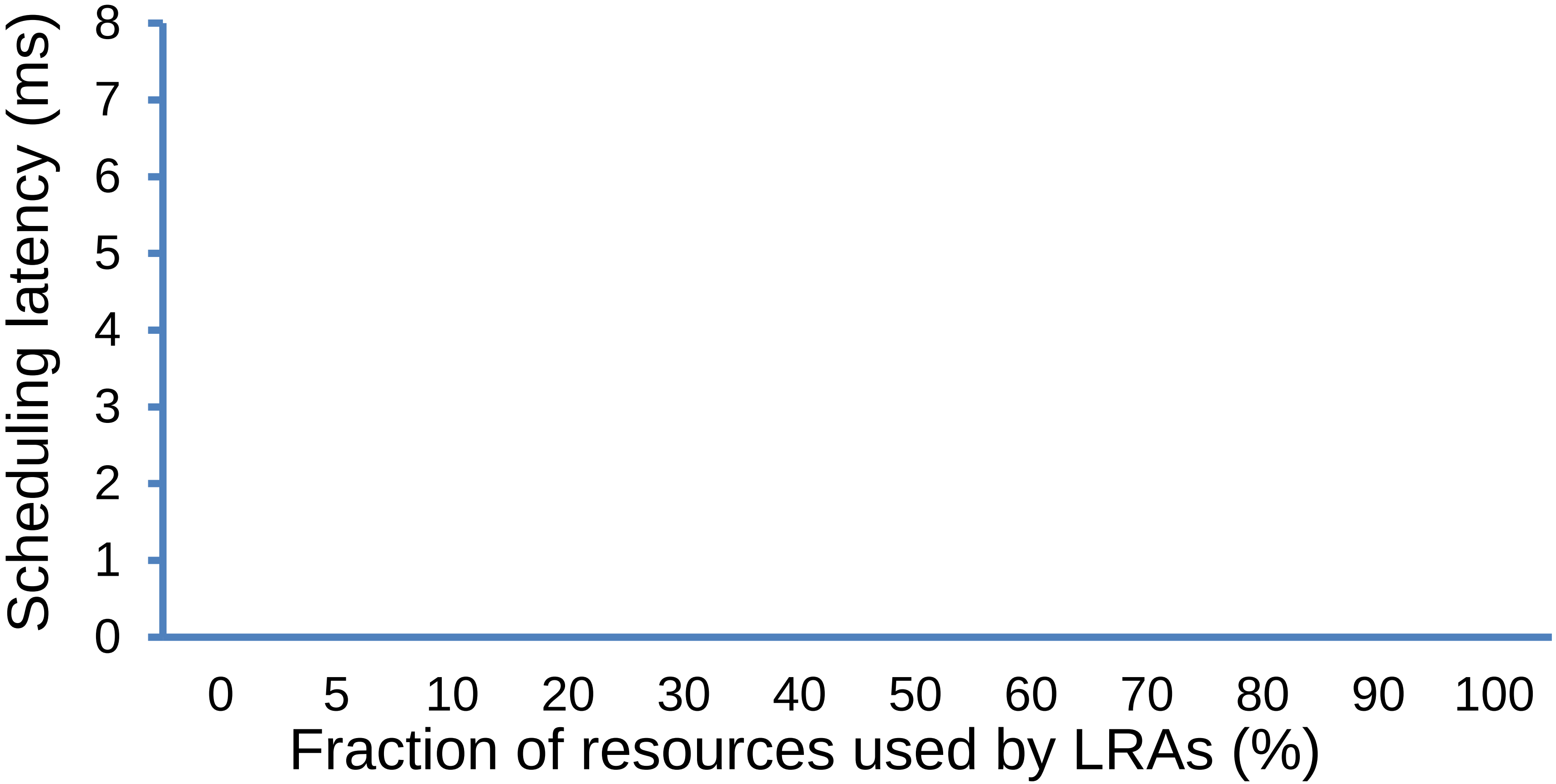
Affinity and anti-affinity are not enough

Impact of cardinality in LRA performance

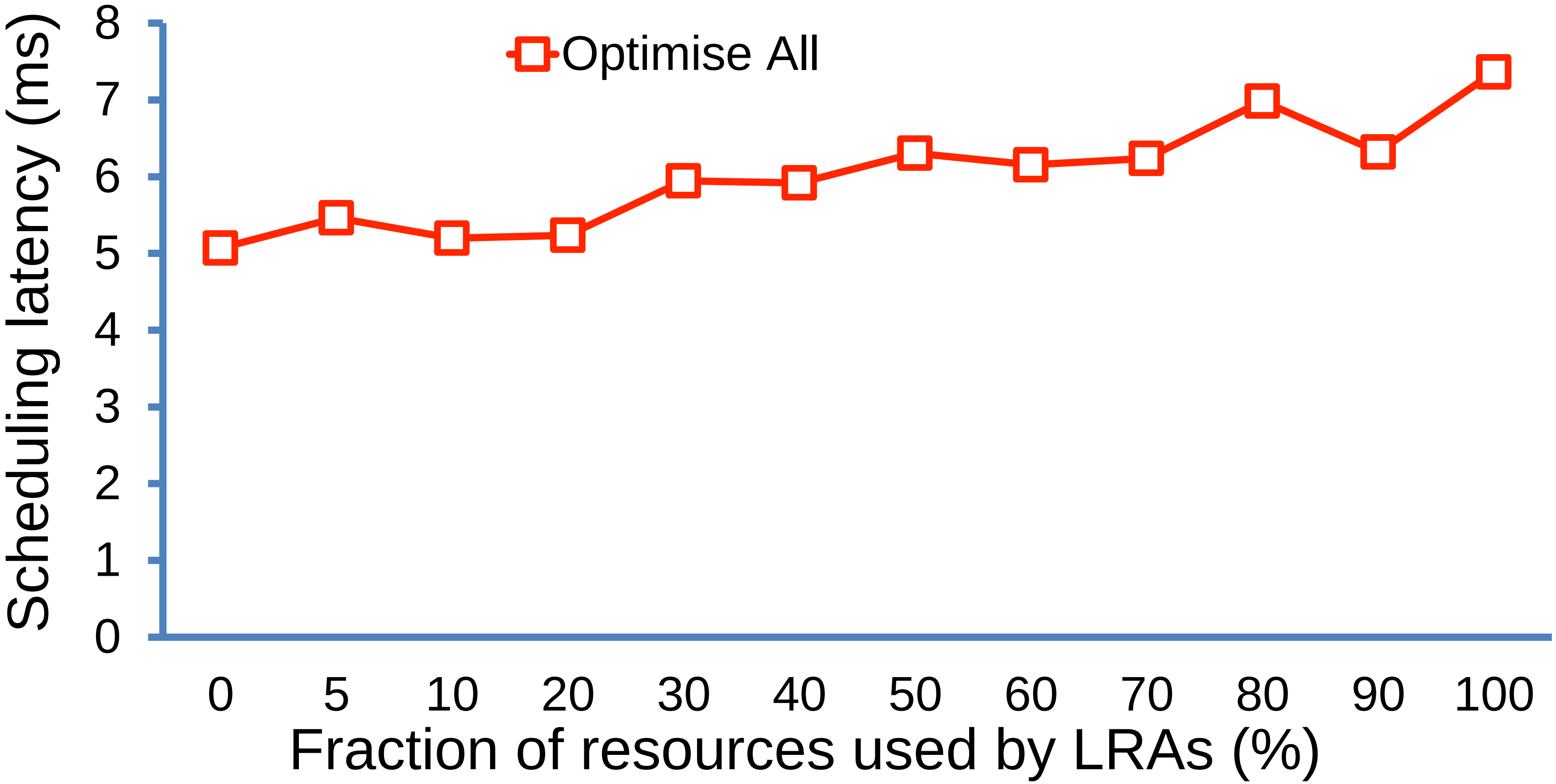


Cardinality constraints are important
Affinity and anti-affinity are not enough

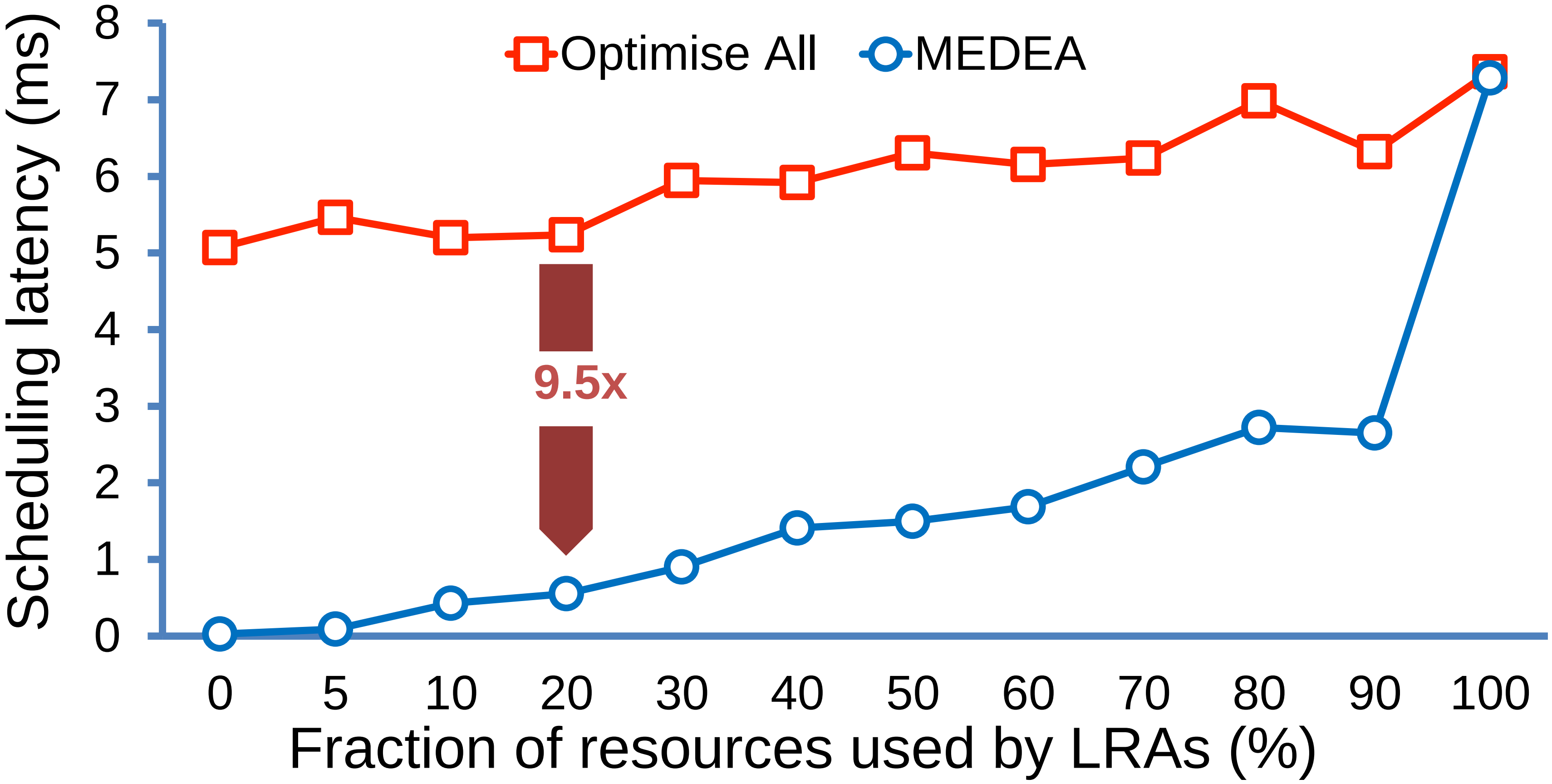
Two-scheduler benefit



Two-scheduler benefit



Two-scheduler benefit



Expensive scheduling logic is used where it matters the most

Large scale deployment

> Pre-production cluster

- 400 machines on 10 racks

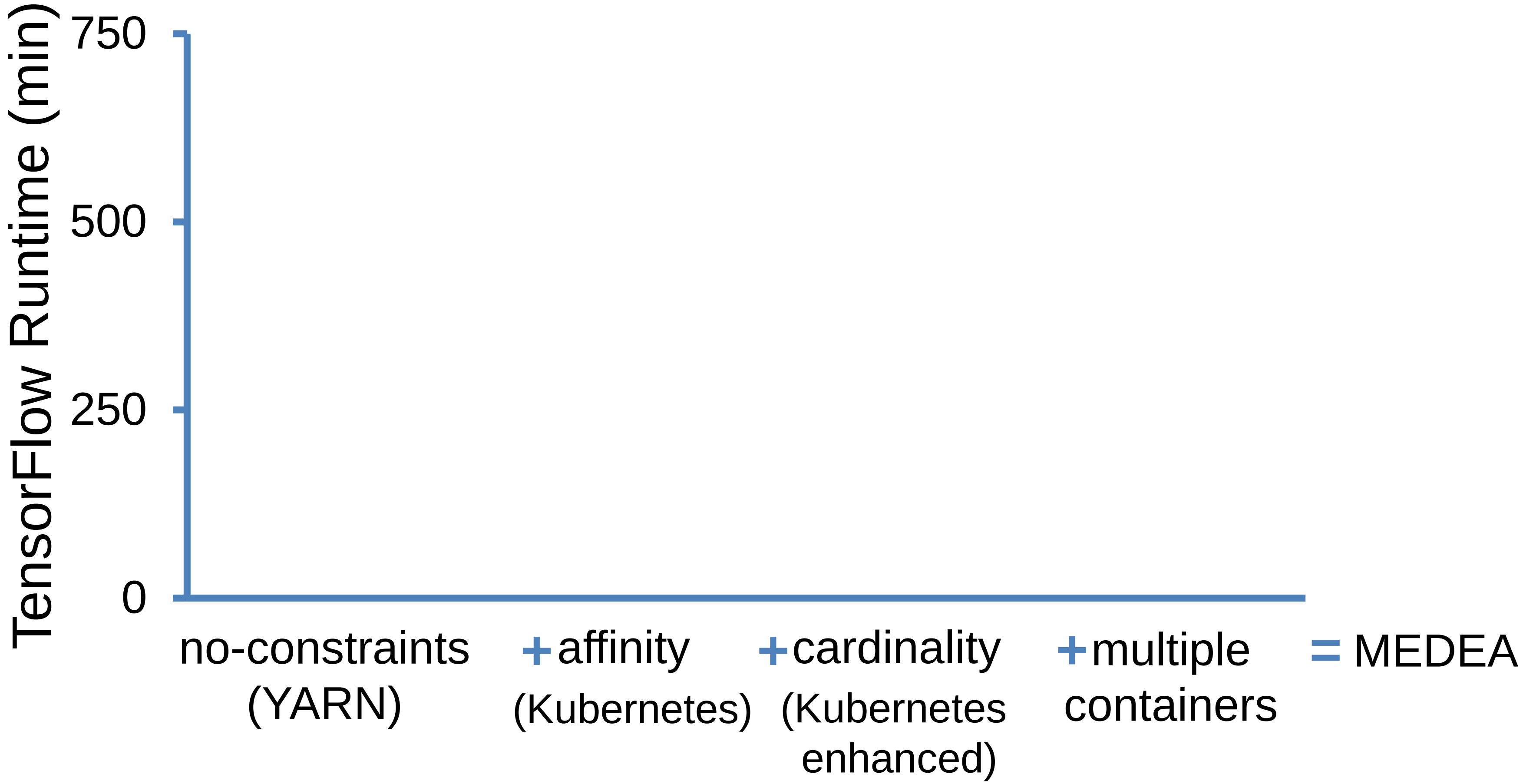
> Workloads

- 50 HBase instances (10 workers each)
- 45 TensorFlow instances (8 workers and 2 PS each)
- Batch production workload (50% of cluster resources)

> Constraints

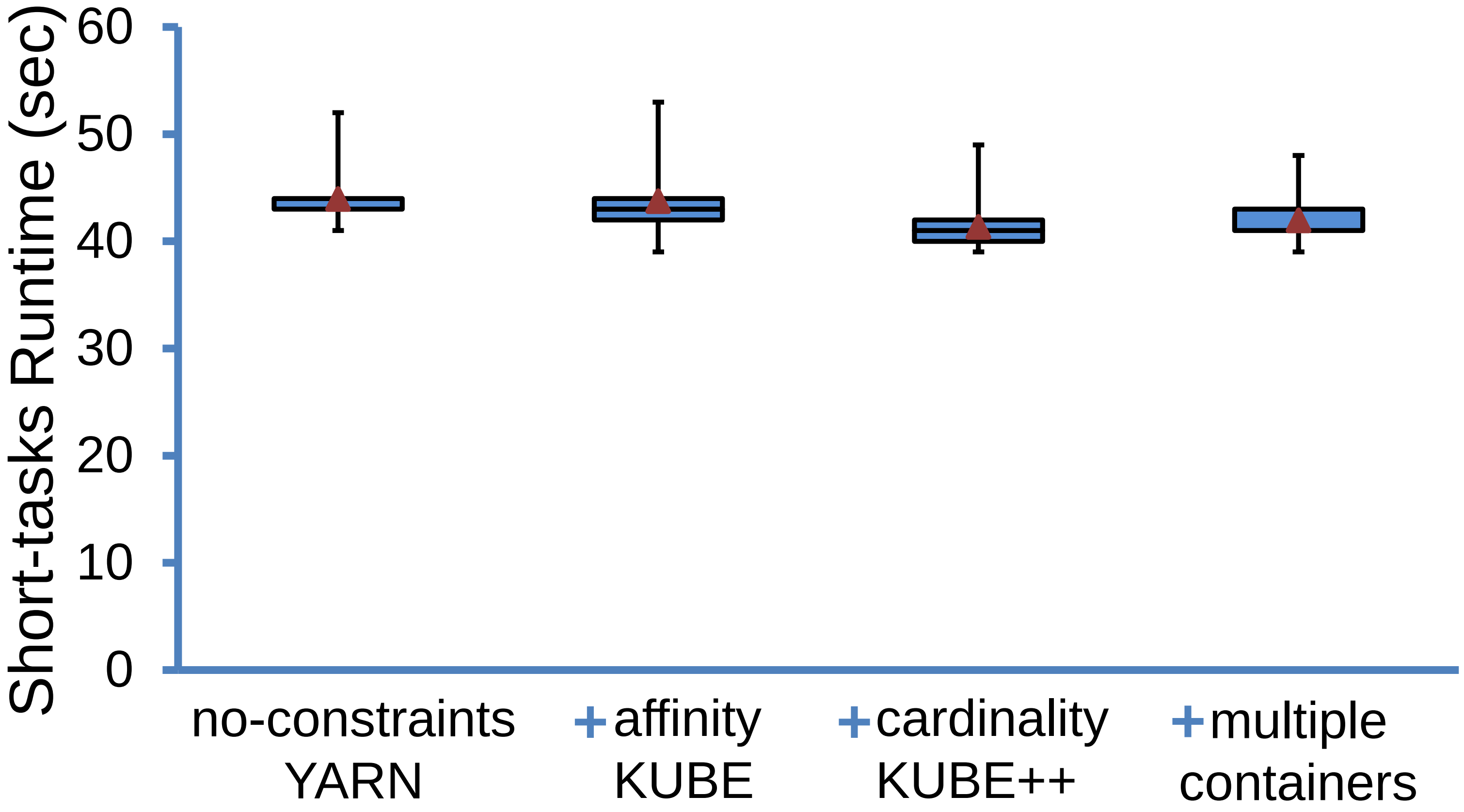
- Containers of each LRA instance on the same rack
- No more than 2 HBase (4 for TensorFlow) containers on same node

Impact of MEDEA in LRA performance



MEDEA improves performance and predictability of LRAs

Impact of MEDEA in Task performance



MEDEA does not affect task-based job runtimes

Summary

Powerful constraints are required to unlock the full potential of LRAs!

MEDEA

- – Two-scheduler design
- – Support expressive & high level constraints
- – Does not impact latency of task-based jobs

APACHE hadoop

YARN r3.1.0

https://github.com/apache/hadoop

Thank you!
Questions?

Panagiotis Garefalakis
pg1712@imperial.ac.uk