MOHA: Many-Task Computing Framework on Hadoop

Soonwook Hwang

Korea Institute of Science and Technology Information

May 18, 2017

KiSTi
www.kisti.re.kr

Table of Contents

Introduction

- **Distributed/Parallel computing systems to support various types of challenging applications**
  - **HTC (High-Throughput Computing)**
    - ✓ a computing paradigm focusing on the efficient execution of a large-number of *loosely-coupled* tasks (e.g., independent and sequential tasks) over long periods of time
    - ✓ exploiting many optimistic computing resources deployed across multiple distributed administration domains
    - ✓ primary metrics: # of operations or tasks per month

BOINC HTCondor High Throughput Computing SETI HOME HTCaaS

  - **HPC (High-Performance Computing)**
    - ✓ targets efficiently processing *tightly-coupled parallel* tasks usually being executed on a particular site (e.g., on a supercomputer) with low-latency interconnects
    - ✓ primary metrics: # of floating point operations per second (FLOPS)

MPI

Introduction

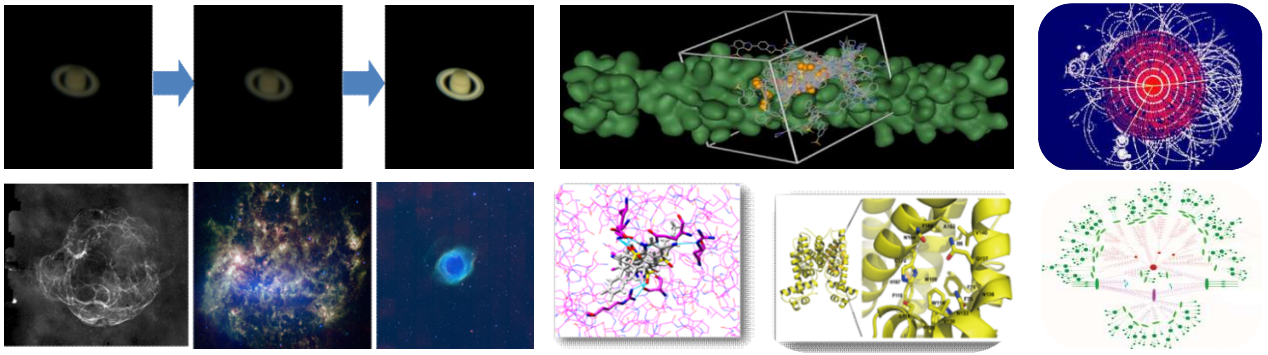
- **DIC (Data-intensive Computing)**
  - A parallel processing framework employing data parallel approach to efficiently process large volumes of data (e.g., terabytes or petabytes in size typically referred to as big data)
  - Devoting most of its processing time on I/O and movement and manipulation of data

- **CIC (Compute-intensive computing)**
  - Used to describe compute-bound applications
  - Devoting most of its execution time to computational requirements as apposed to I/O (e.g., typically requiring small volume of data)

MPI

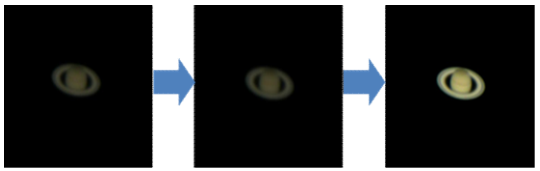
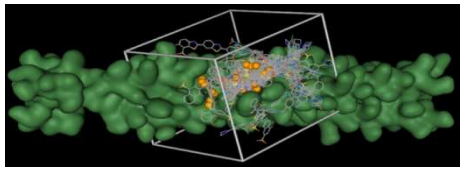
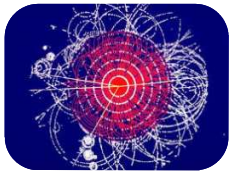
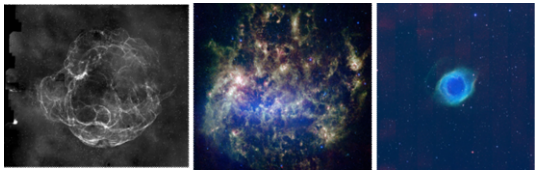
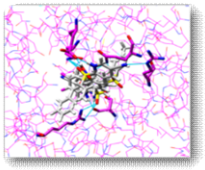
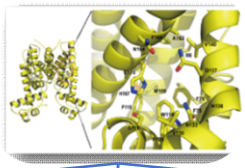
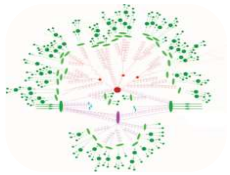
Introduction

- **Many-Task Computing (MTC) as a *new computing paradigm* [I. Raicu, I. Foster, Y. Zhao, MTAGS'08]**
  - A very **large** number of tasks (millions or even billions)
  - Relatively **short** per task execution time (seconds to minutes)
  - A large **variance** of task execution time (i.e., ranging from hundreds of milliseconds to hours)
  - **Data** intensive tasks (i.e., lots of relatively small-sized file operations ranging from hundreds of KBs to tens of MBs of I/Os)
  - Communication-intensive, however, not based on message passing interface but rather through **file operations.**



astronomy, physics, pharmaceuticals, chemistry, etc.

Introduction

Many-Task Computing Applications

astronomy, physics, pharmaceuticals, chemistry, etc.

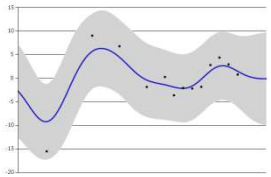
A very **large** # of tasks

millions or even billions

Relatively **short** pertask execution time

seconds to minutes

A large **variance** of task execution time



from hundreds of milliseconds to hours

Data intensive tasks

tens of MB of I/O per second

Communication through **files**

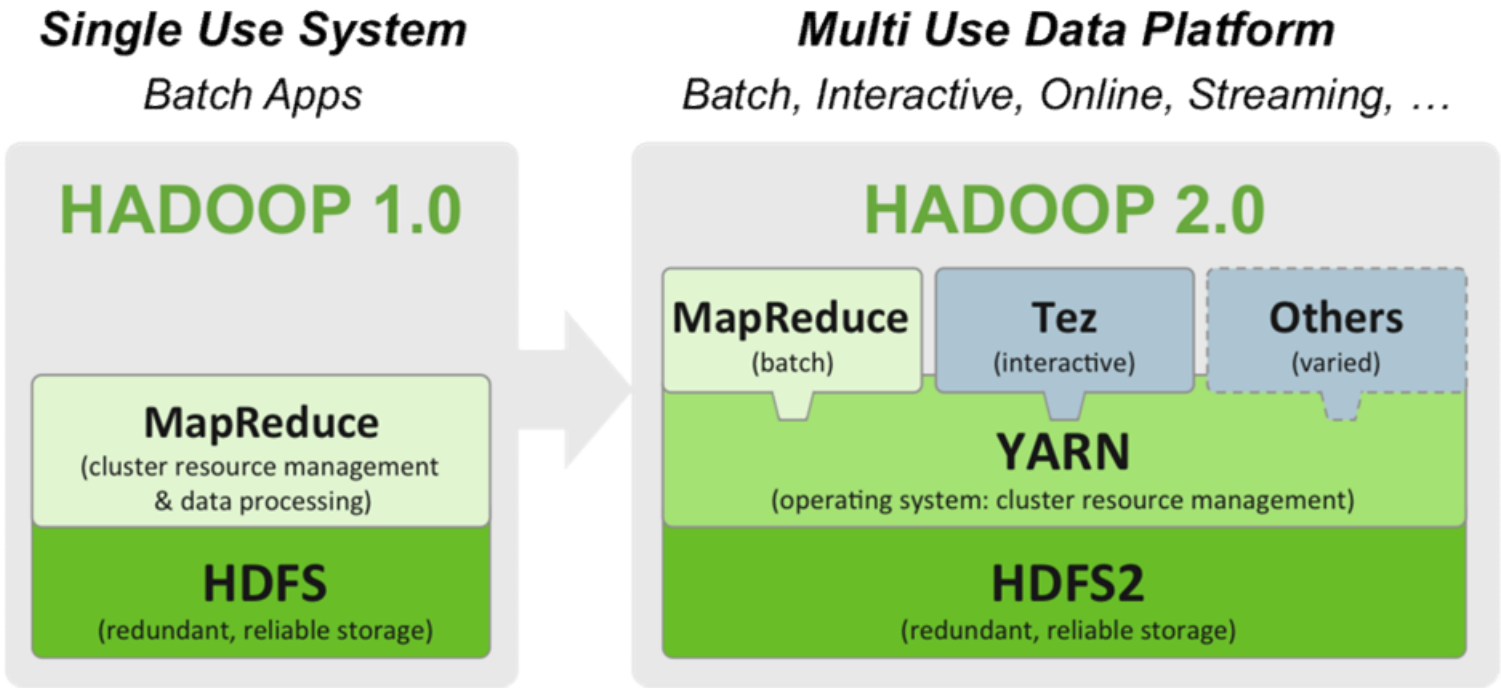
High-Performance Task Dispatching

Dynamic Load Balancing among compute nodes

***Another Type* of Data-intensive Workload**

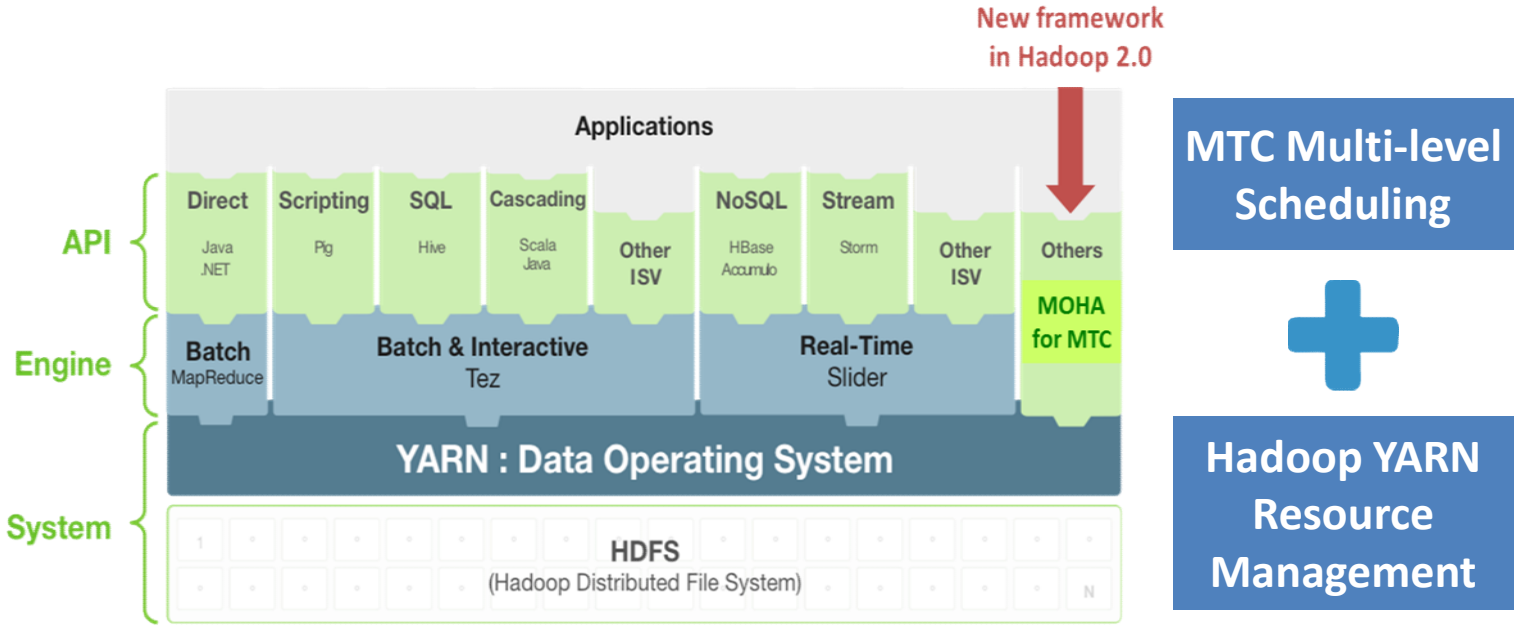
Introduction

- **Hadoop, the *de facto standard* "Big Data" store and processing infrastructure**
  - with the advent of **YARN**, Hadoop is moving beyond MapReduce and Batch processing, and evolving into **multi-use data platform**
    - ✓ build/run multiple distributed frameworks/applications on top of Hadoop cluster



Introduction

- **YARN facilitates the development of other purpose-built data processing models beyond MapReduce**
- **MOHA (Many-task computing On HAdoop):**
  - Proof of Concept development for enabling the Many-Task Computing practices on a Hadoop Cluster
  - currently prototyped as a YARN application



Related Work

- **GERBIL: MPI+YARN [L. Xu , M. Li, A. R. Butt, CCGrid'15]**
  - A framework for hosting MPI applications on a Hadoop cluster so as to transparently co-host unmodified **MPI applications** alongside MapReduce applications
    - ✓ allows realization of **rich data analytics workflows** as well as efficient data sharing between the MPI and MapReduce models within a single cluster
  - Scientific applications like Metagenomics are beneficial from GERBIL
    - ✓ Consisting of compute and communication-intensive tasks (MPI) and data-intensive tasks (MapReduce) in its workflow

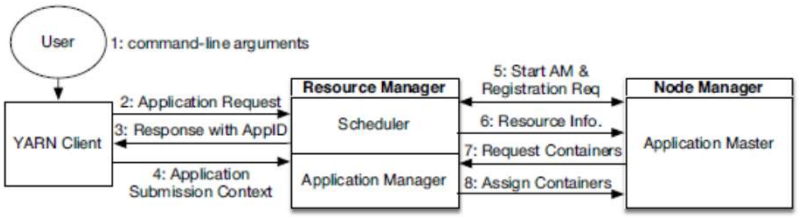


Fig. 1. Steps of launching an application in YARN.

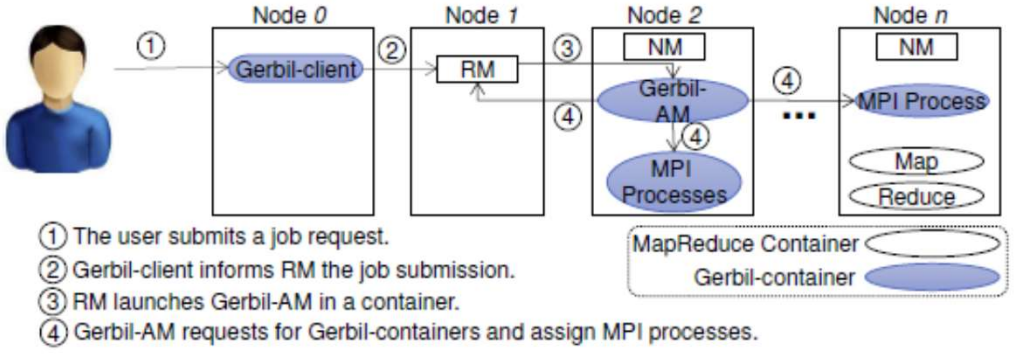
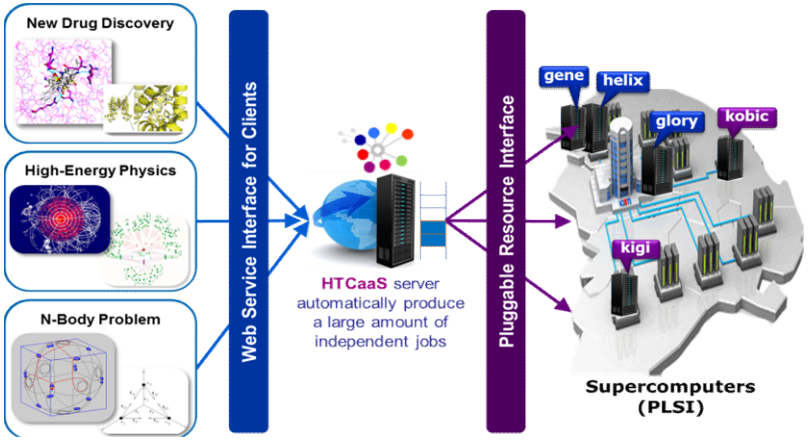


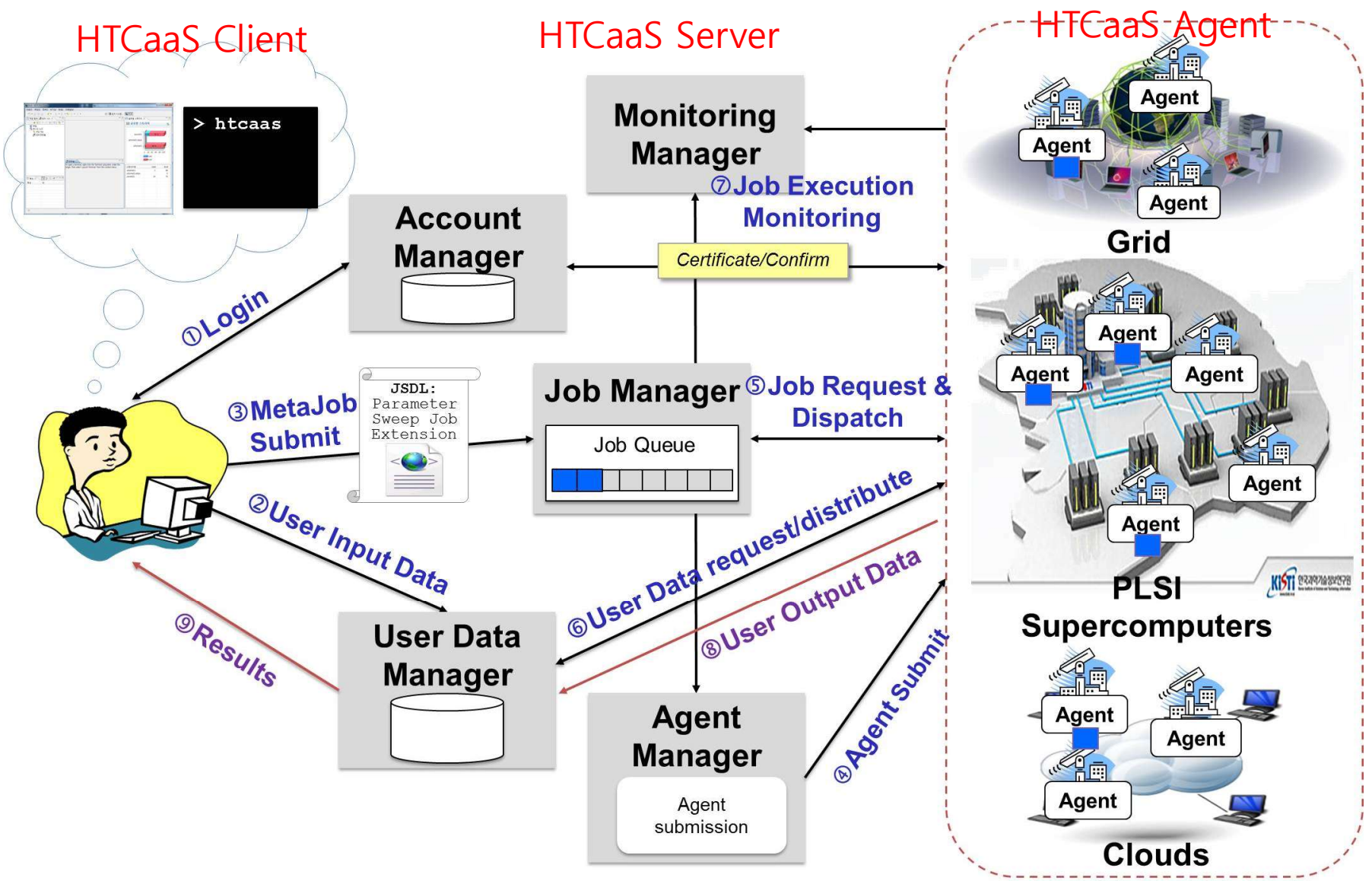
Fig. 2. GERBIL architecture for running MPI on YARN.

Related Work: HTCaaS

- **We have developed the HTCaaS system for MTC execution on top of a distributed & heterogeneous computing infrastructure (e.g., Clusters, Supercomputers, Grids, etc)**
- **HTCaaS: a Multi-level Scheduling System**
  - *flexible and powerful* high-level job description in XML
    - ✓ e.g., parameter sweeps or N-body calculations
  - automated job split into a set of tasks to be enqueued
  - agent-based *multi-level scheduling*
    - ✓ *1st level scheduler for launching agents and 2nd level scheduler for task pulling & execution and handling fault tolerances etc.*
  - *data staging-in & staging-out*
  - HTCaaS is currently running as a **pilot service** on top of PLSI
    - ✓ supporting a number of scientific applications from **pharmaceutical domain** and **high-energy physics**



Related Work: HTCaaS



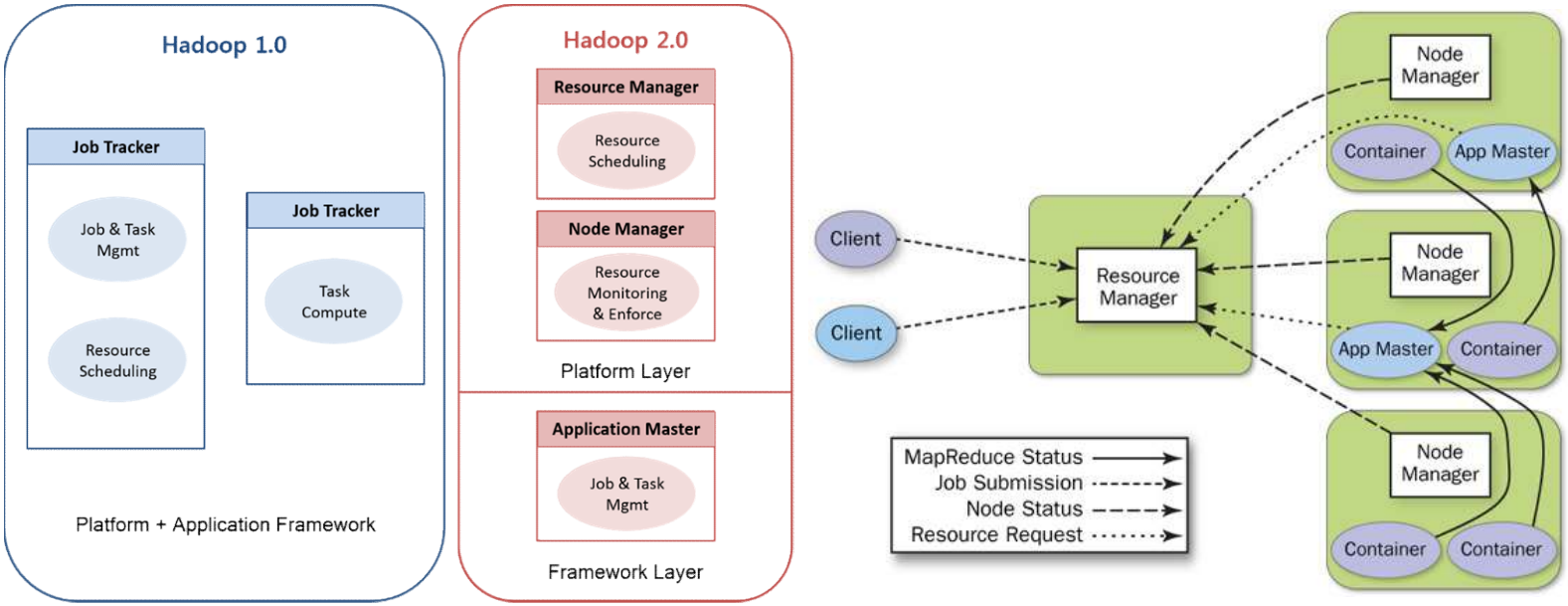
YARN vs. HTCaaS

	YARN	HTCaaS
Platform Layer (1st level scheduling)	Resource Manager (RM)	HTCaaS Server (e.g., HTCaaS Agent Manager Component)
	Node Manager (NM)	Local Resource Manager running on local data/computing sites (e.g., PBS, Grid RM, Cloud RM)
Application Framework Layer (2nd level scheduling)	YARN Client	HTCaaS Client
	YARN Application Master	HTCaaS Server
	YARN Container	HTCaaS Agent

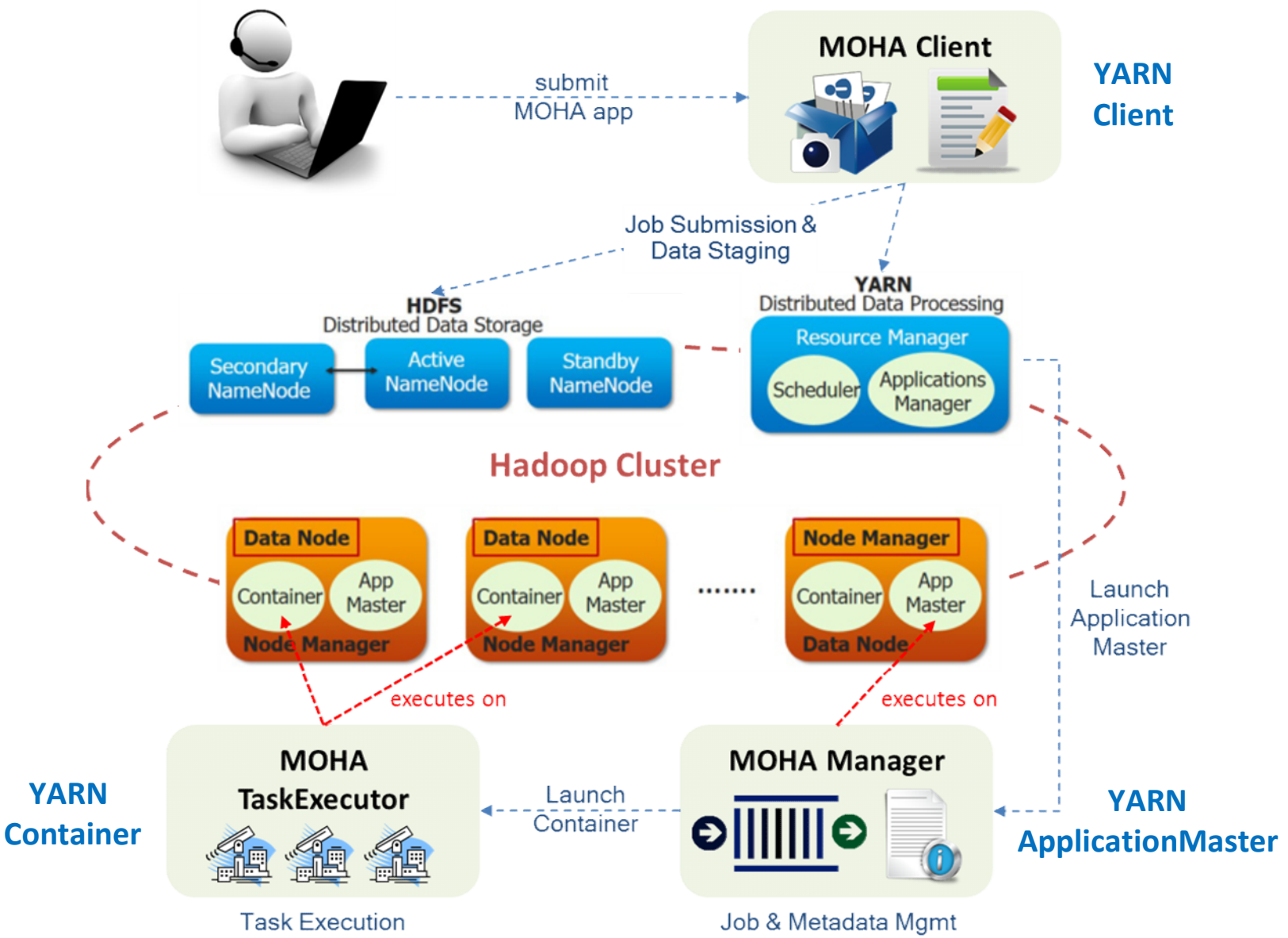
Table of Contents

Hadoop YARN Execution Model

- **YARN separates all of its functionality into *two* layers**
  - **platform layer** is responsible for resource management (*first-level scheduling*)
    - ✓ **Resource Manager, Node Manager**
  - **framework layer** coordinates application execution (*second-level scheduling*)
    - ✓ **Client, ApplicationMaster, Container**

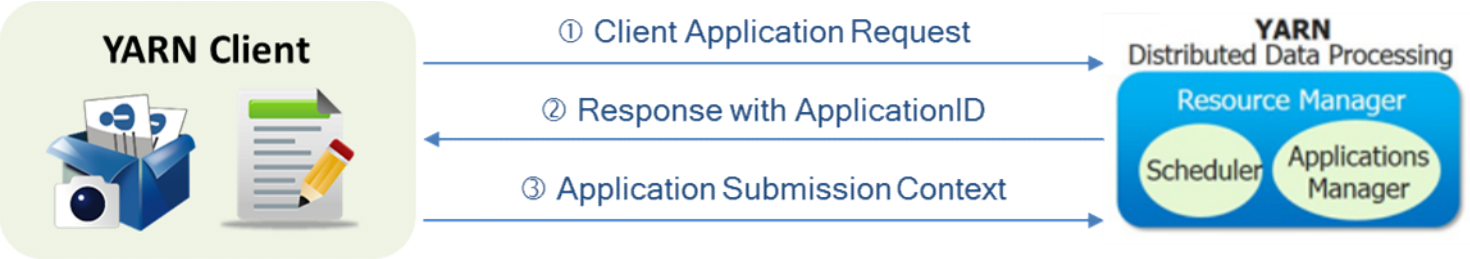


MOHA System Architecture



MOHA System Architecture

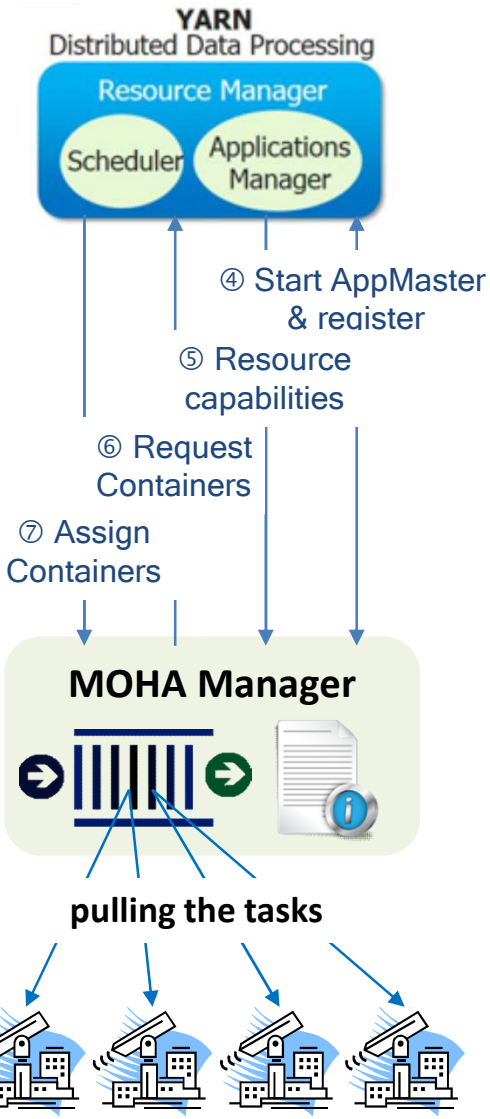
- **MOHA Client**
  - **create & submit a MOHA MTC job** and performs **data staging**
    - ✓ A MOHA job is *a bag of tasks* (i.e., a collection of multiple tasks)
      - provides a simple JDL(Job Description Language)
    - ✓ upload required data into the HDFS
      - application input data, application executable, MOHA JAR, JDL etc.
  - prepare an execution environment for the MOHA Manager based on *YARN's Resource Localization Mechanism*
    - ✓ required data are *automatically* downloaded and prepared for use in the local working directories



MOHA System Architecture

- **MOHA Manager**
  - create a **MOHA job queues**

    Apache Kafka — A high-throughput distributed messaging system. ActiveMQ
  - **split** a MOHA MTC job described in JDL into multiple tasks and insert them into the queue
  - get containers **allocated** and **launch** MOHA TaskExecutors
- **MOHA TaskExecutor**
  - **pull** the MOHA tasks from the MOHA job queues and process them
    - ✓ monitor and report the task execution



"Multi-level Scheduling Mechanism"

Table of Contents

Experimental Setup

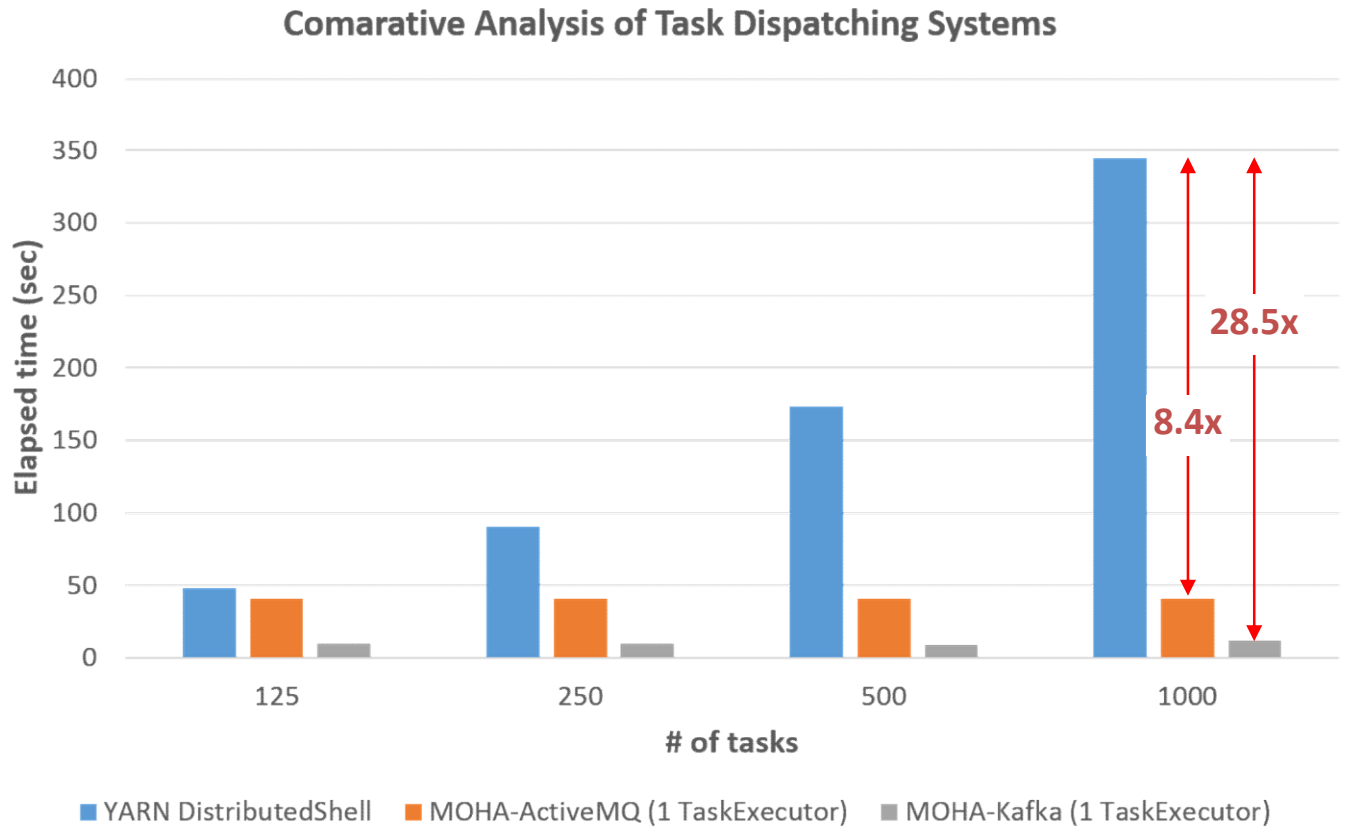
- **MOHA Testbed**
  - consists of 3 rack mount servers
    - ✓ 2 * Intel Xeon E5-2620v3 CPUS (12 CPU cores)
    - ✓ 64GB of main memory
    - ✓ 2 * 1TB SATA HDD (1 for Linux, 1 for HDFS)
  - Software stack
    - ✓ **Hortonworks Data Platform (HDP) 2.3.2**
      - automated install with Apache **Ambari**
    - ✓ Operating Systems Requirements
      - CentOS release 6.7 (Final)
    - ✓ Identical environment with the **Hortonworks Sandbox VM**

Experimental Setup

- **Comparison Models**
  - **YARN Distributed-Shell**
    - ✓ a simple YARN application that can execute shell commands (scripts) on distributed containers in a Hadoop cluster
  - **MOHA-ActiveMQ**
    - ✓ ActiveMQ running on a single node with New I/O (NIO) Transport
  - **MOHA-Kafka**
    - ✓ 3 Kafka Brokers with minimum fetch size (64 bytes)
- **Workload**
  - Microbenchmark
    - ✓ varying the # of “sleep 0” tasks
  - Performance Metrics
    - ✓ Elapsed time (i.e., makespan or turnaround time)
    - ✓ Task processing rate (# of tasks/sec)

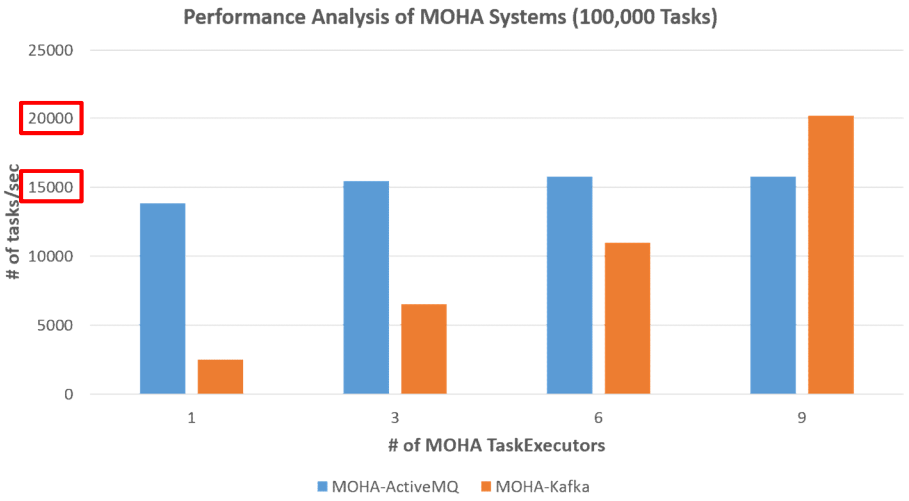
Experimental Results

- **Performance Comparison (Total Elapsed Time)**
  - multiple resource (de)allocations in YARN Distributed-Shell
  - multi-level scheduling mechanisms enable MOHA frameworks to substantially reduce the cost of executing many tasks



Experimental Results

- **Task Dispatching Rate and Initialization Overhead**
  - MOHA-Kafka outperforms MOHA-ActiveMQ as the number of TaskExecutors increases (also Falkon's 15,000 tasks/sec)
    - ✓ have not fully utilized Kafka's task bundling functionality
  - Initialization Overhead
    - ✓ mostly queuing time



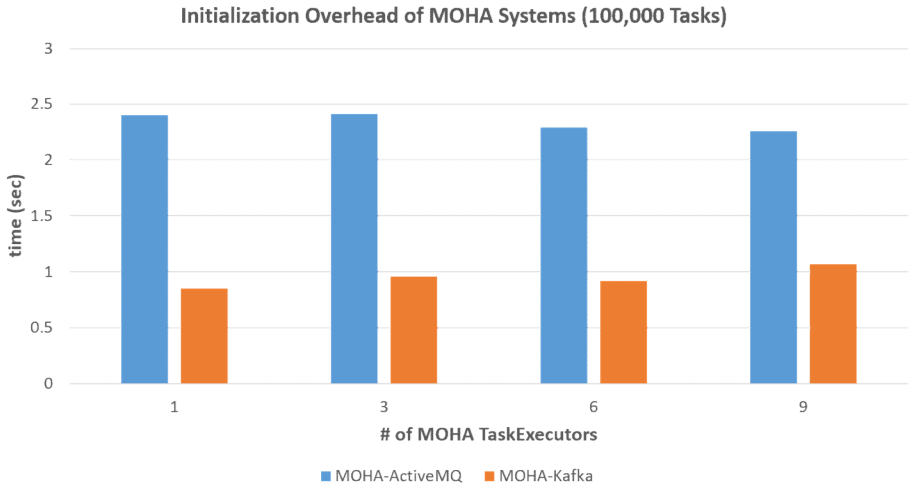


Table of Contents

Discussion

- **MTC applications typically require**
  - much *larger* numbers of tasks
  - relatively *short* per-task execution time
  - substantial amount of *data* operations with potential interactions through *files*

  ☑ high-performance task dispatching

  ☑ effective dynamic load balancing

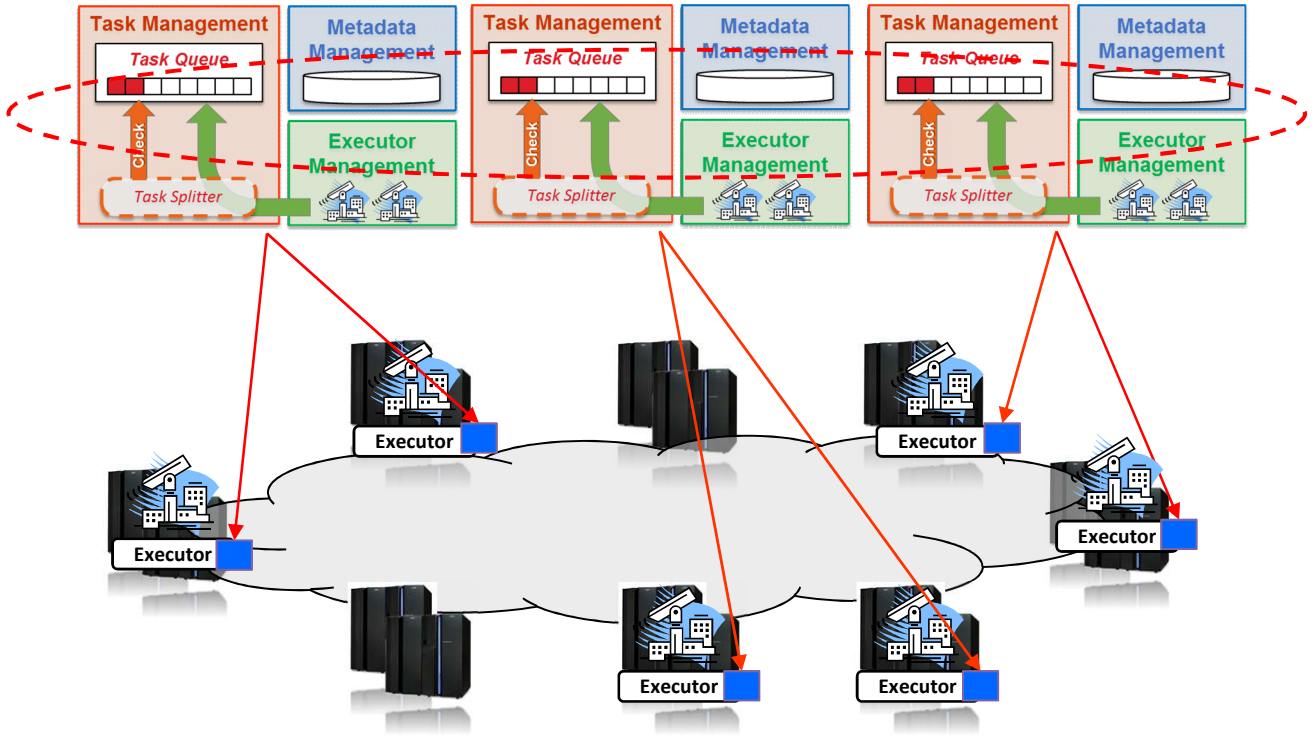
  ☑ data-intensive workload support

  ⇨ **"seamless integration"**
- **Hadoop can be a viable choice for addressing these challenging MTC applications**
  - technologies from MTC community should be effectively *converged* into the ecosystem

Discussion

Potential Research Issues

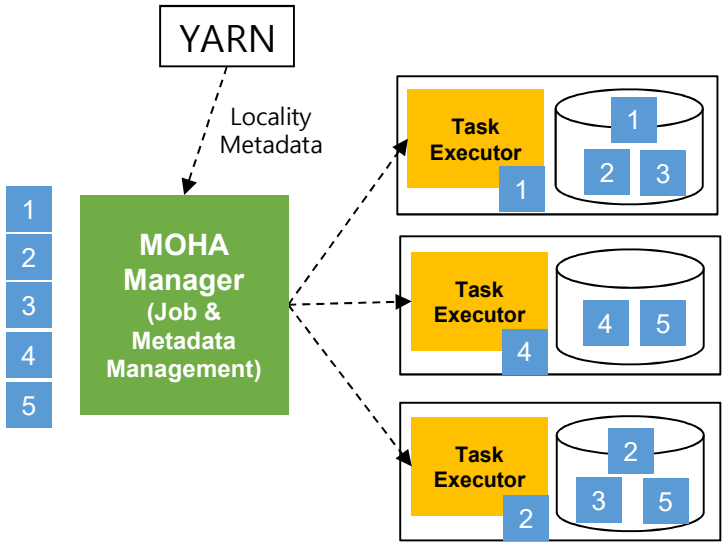
- Scalable Job/Metadata Management
  - ✓ removing potential performance bottleneck
- Dynamic Task Load Balancing
  - ✓ Task bundling and Job profiling techniques

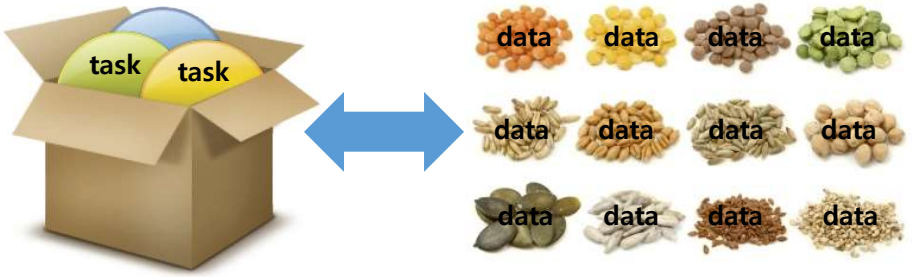


Discussion

Potential Research Issues

- Data-aware resource allocation
  - ✓ leveraging Hadoop's data locality (computations *close* to data)
- Data Grouping & Declustering
  - ✓ aggregating a groups of small files ("data bundle")





Task Bundling & Data Grouping can be closely related

Table of Contents

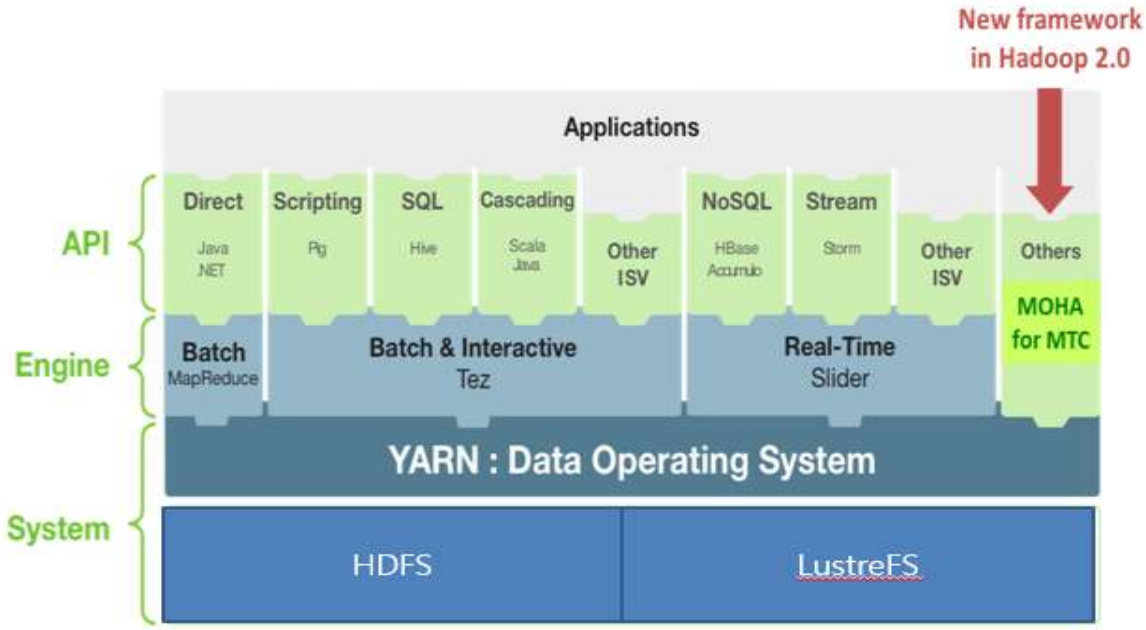
Conclusion

- **Design and implementation of MOHA (Many-task computing On HAdoop) framework**
  - effectively *combine* MTC technologies with Hadoop
  - developed as *one of Hadoop YARN applications*
  - transparently *co-host* existing MTC applications with other Big Data processing frameworks in a single Hadoop cluster
- **MOHA prototype as a Proof-of-Concept**
  - can execute *shell* command based many tasks across distributed computing resources
  - substantially *reduce* the overall execution time of many-task processing with *minimal* amount of resources
    - ✓ compared to the existing YARN Distributed-Shell
  - efficiently *dispatch* a large number of tasks by exploiting *multi-level scheduling* and *streamlined task dispatching*

Another work-in-progress PoC development in relevance with the MOHA development

Making Hadoop Ecosystem available for use on top of Lustre file system, too

- **Motivation**
  - MTC often requires a large number (e.g., tens of millions) of relatively small-sized (e.g., tens or hundreds of KBs to MBs) of file I/O operations over the course of a MTC job execution
    - ✓ doesn't fit with the underlying HDFS where a typical data block size is 64MB
  - Most of scientific data, the major target data that MTC often has to deal with, is stored in parallel file system (PFS) like Lustre.
    - ✓ Requires performance-degrading manual data copying between the two file systems: HDFS and Lustre file system.




Thank you!
National Institute of
Supercomputing and Networking
2017