Foundation of Internet Platform Development & Operation

Review of Homework II

2019-10-29

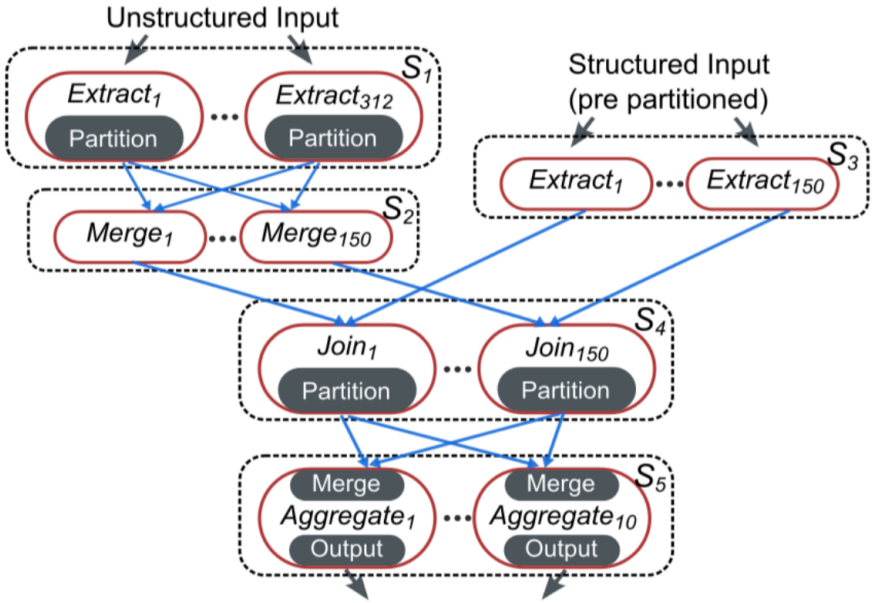
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Apollo

- Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing
- *Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, and Jingren Zhou, Microsoft; Zhengping Qian, Ming Wu, and Lidong Zhou, Microsoft Research*

Apollo

- Workload
  - SQL-like high level language
  - Job query plan is represented as a DAG
  - Task is the the basic computation unit
  - Tasks are grouped in Stages



Apollo

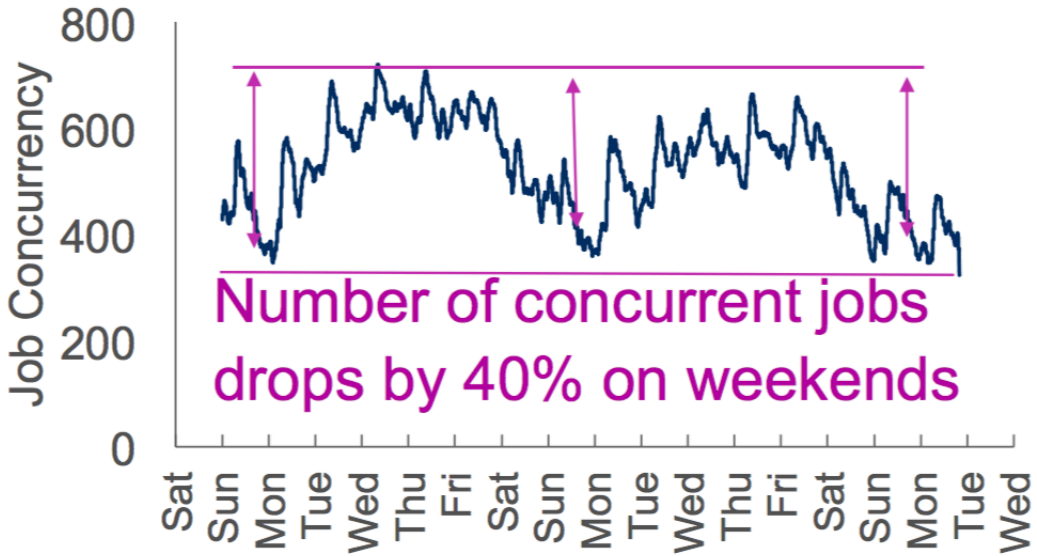
- Challenges
  - Scale
    - Job process GB to PB of data
    - With peaks above 100,000 scheduling requests/second
    - Clusters run up to 170,000 tasks in parallel and each contains over 20,000 servers

Apollo

- Challenges
  - Heterogeneous workload
    - Tasks runs from seconds to hours
    - Tasks can be IO bound or CPU bound
    - Tasks can require from 100MB to more than 10GB of memory
    - Short tasks are sensitive to scheduling latency
    - Long IO bound tasks are sensitive to data locality
  - Maximize utilization
    - How to Maximize utilization while maintaining performance guarantees with a dynamic workload?

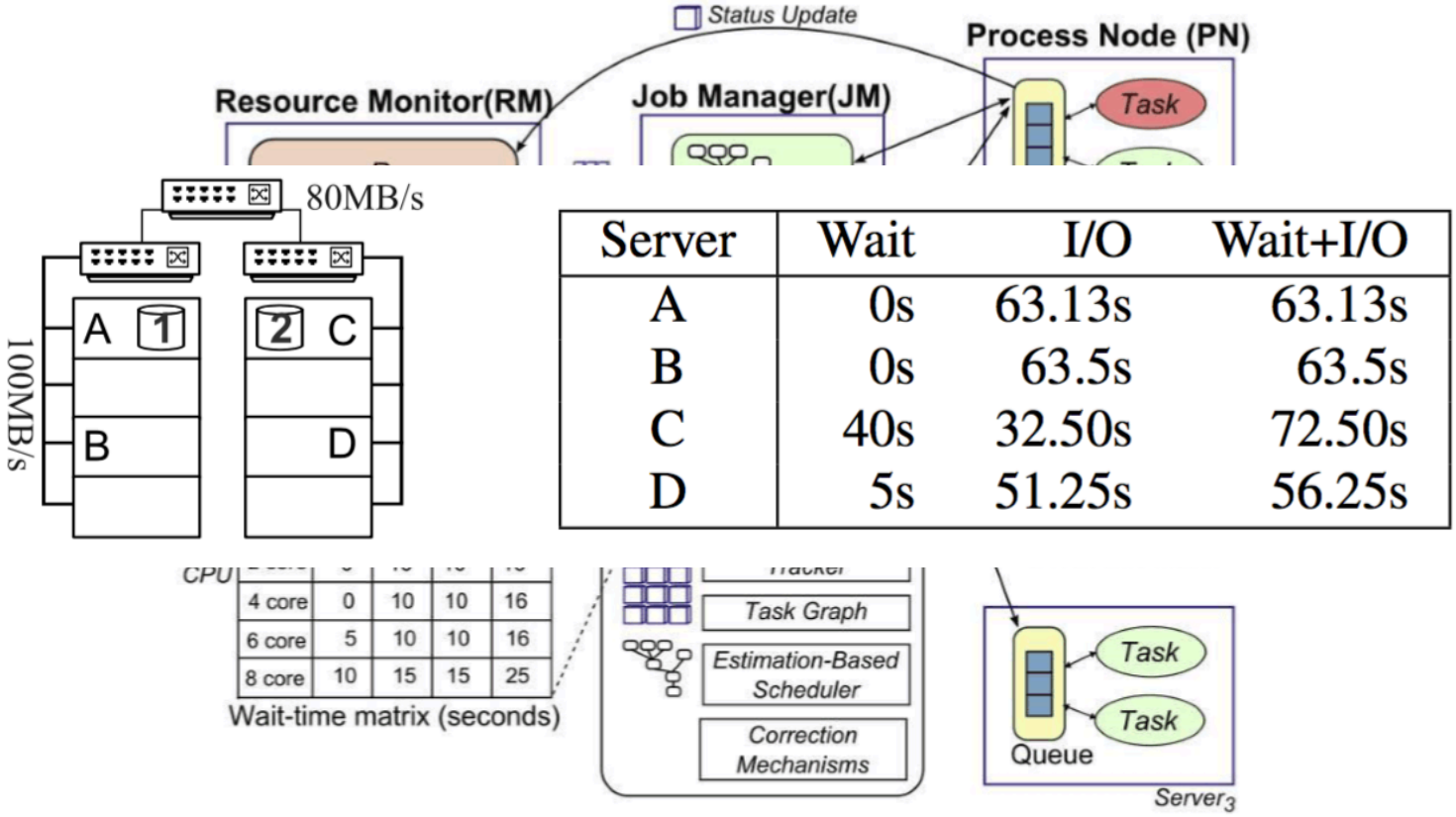
Apollo

- Challenges
  - Maximize utilization
    - How to Maximize utilization while maintaining performance guarantees with a dynamic workload?



Apollo

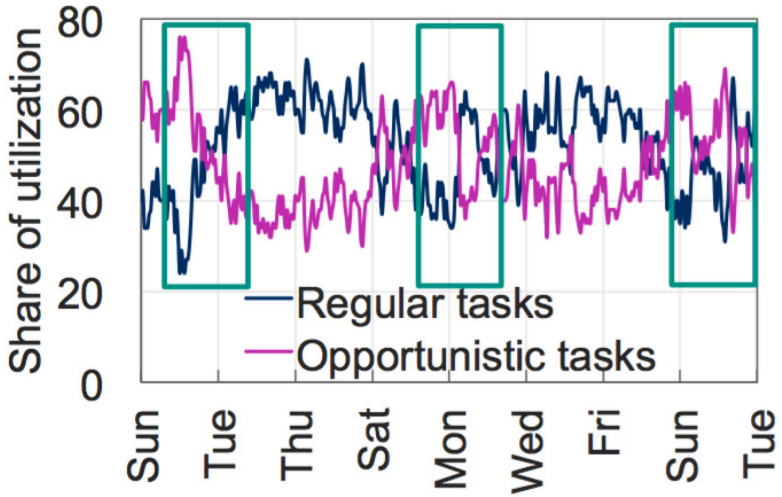
- Distributed and coordinated architecture



Server	Wait	I/O	Wait+I/O
A	0s	63.13s	63.13s
B	0s	63.5s	63.5s
C	40s	32.50s	72.50s
D	5s	51.25s	56.25s

Apollo

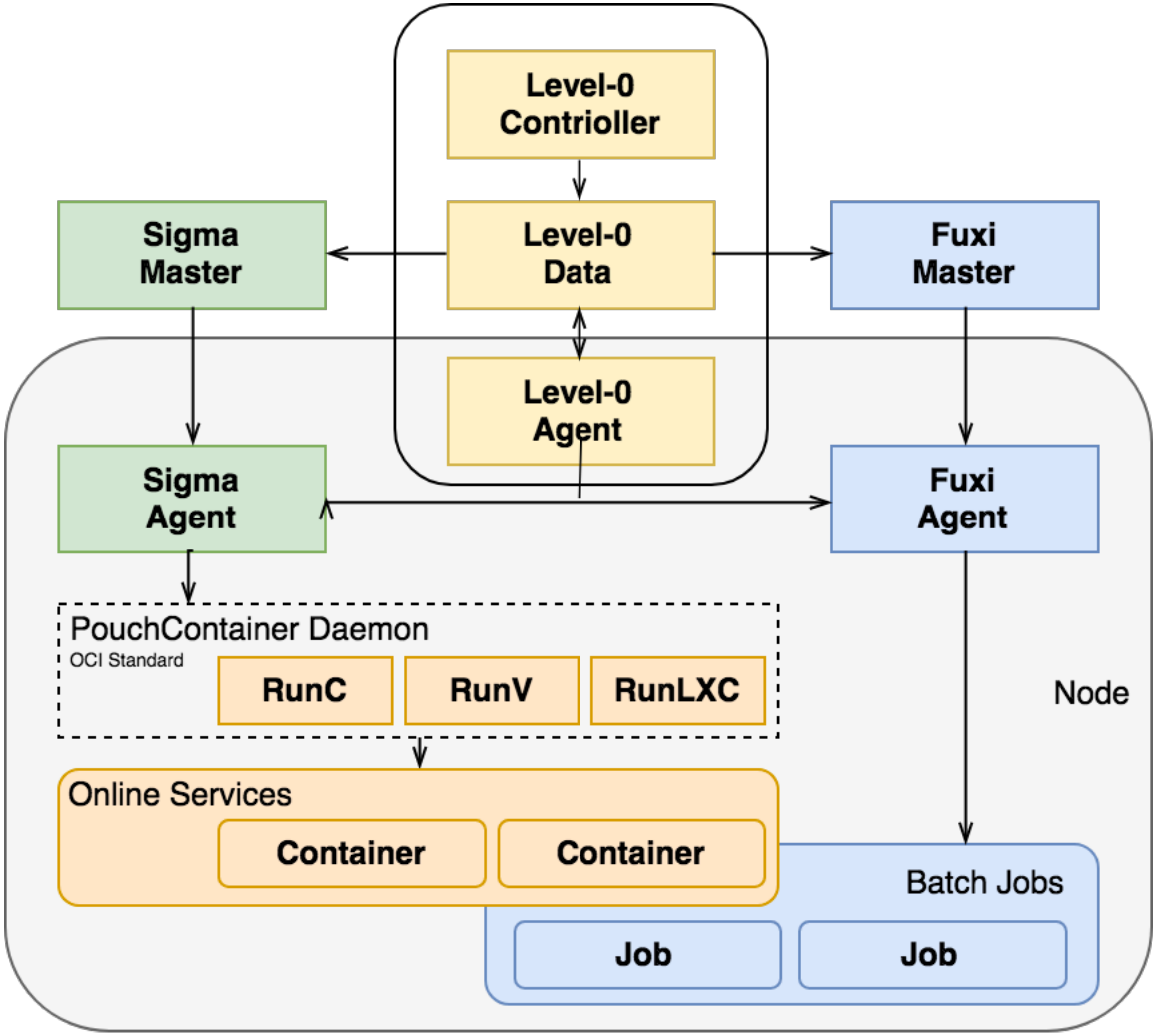
- Opportunistic scheduling
  - Cluster utilization fluctuates over time
  - Idle resources should be utilized whenever available
  - Priority



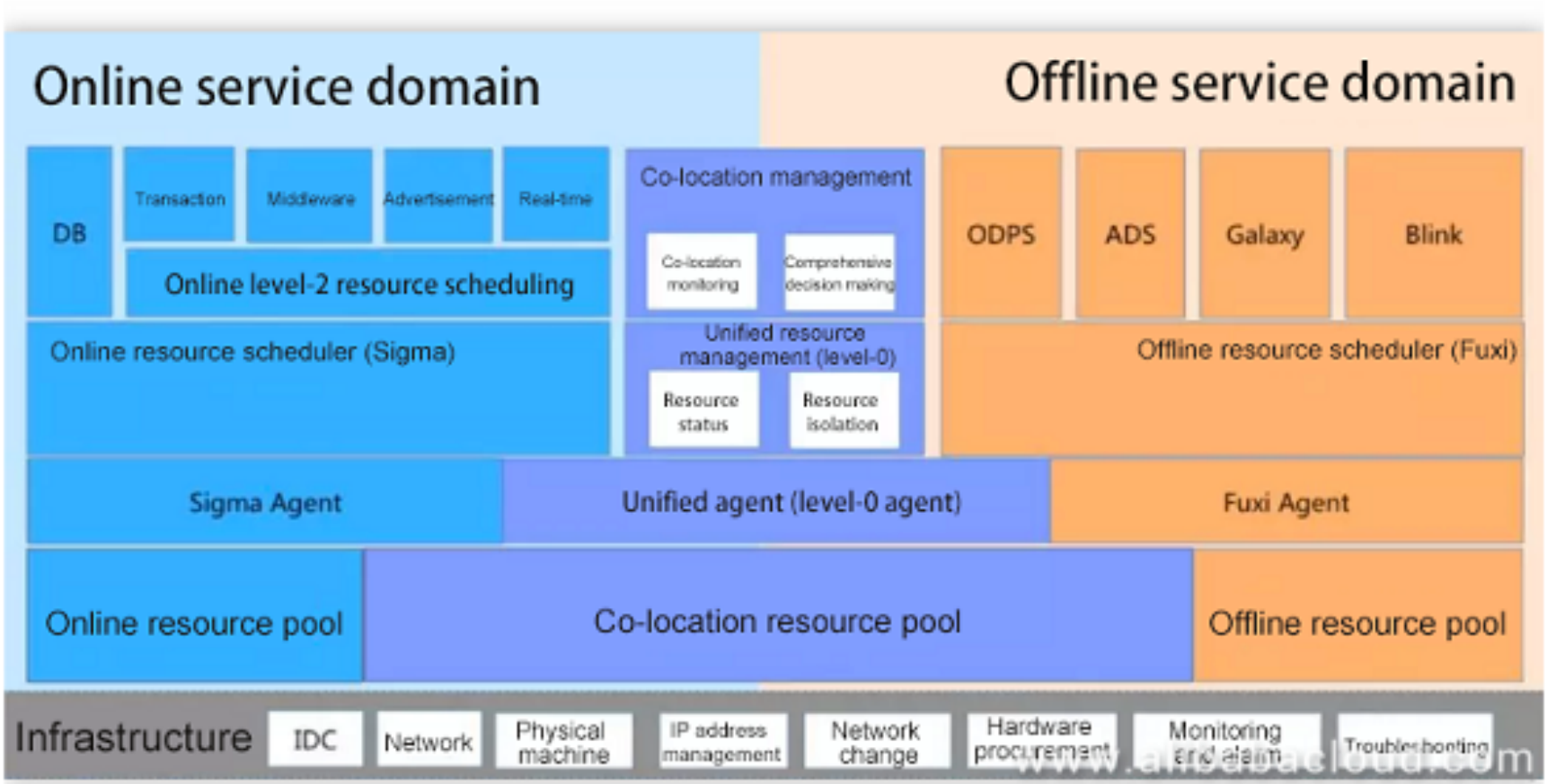
Scheduler Architecture

- Centralized
  - Borg
  - Kubernetes
  - Quincy
- Decentralized
  - Sparrow
  - Omega
- Hierarchical Scheduler
  - Mesos
  - Yarn

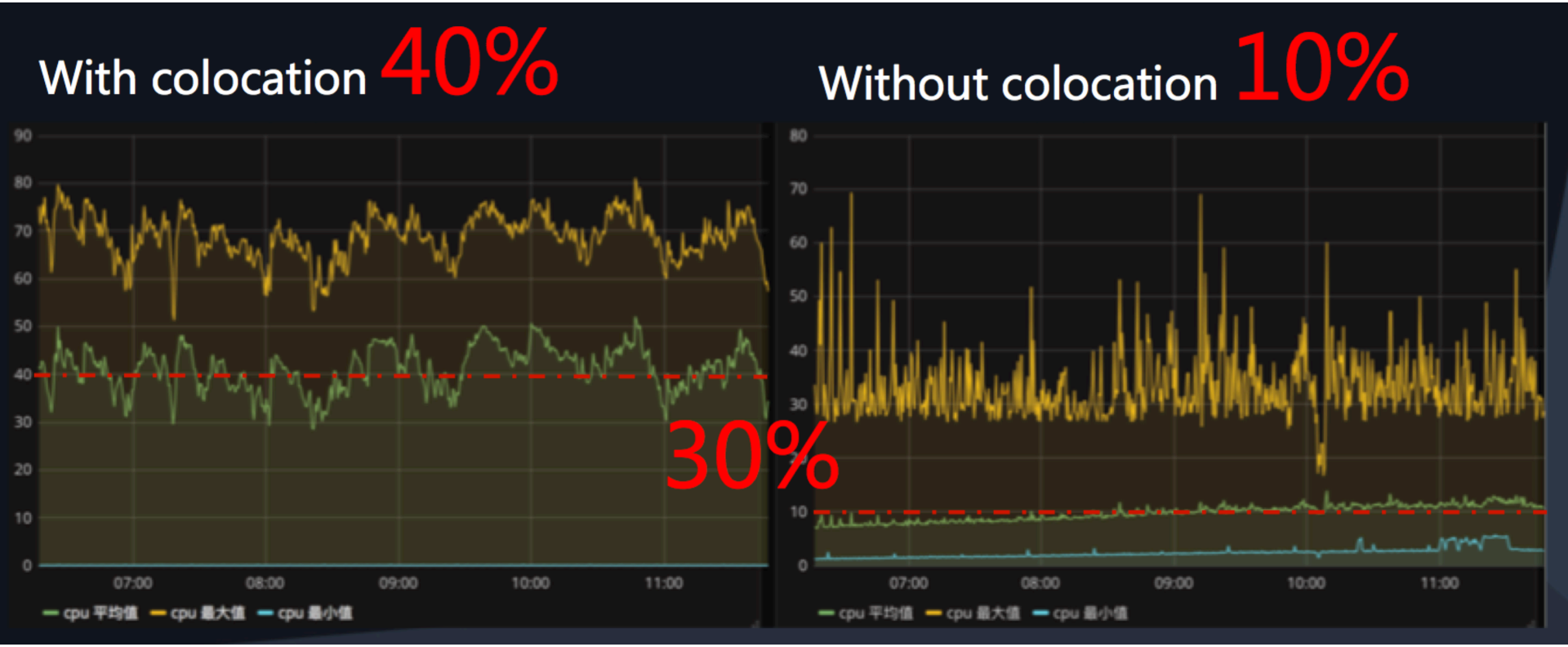
Sigma


Level-0
Contrioller
Level-0
Data
Level-0
Agent
Sigma
Master
Fuxi
Master
Sigma
Agent
Fuxi
Agent
PouchContainer Daemon
OCI Standard
RunC
RunV
RunLXC
Node
Online Services
Container
Container
Batch Jobs
Job
Job

Sigma

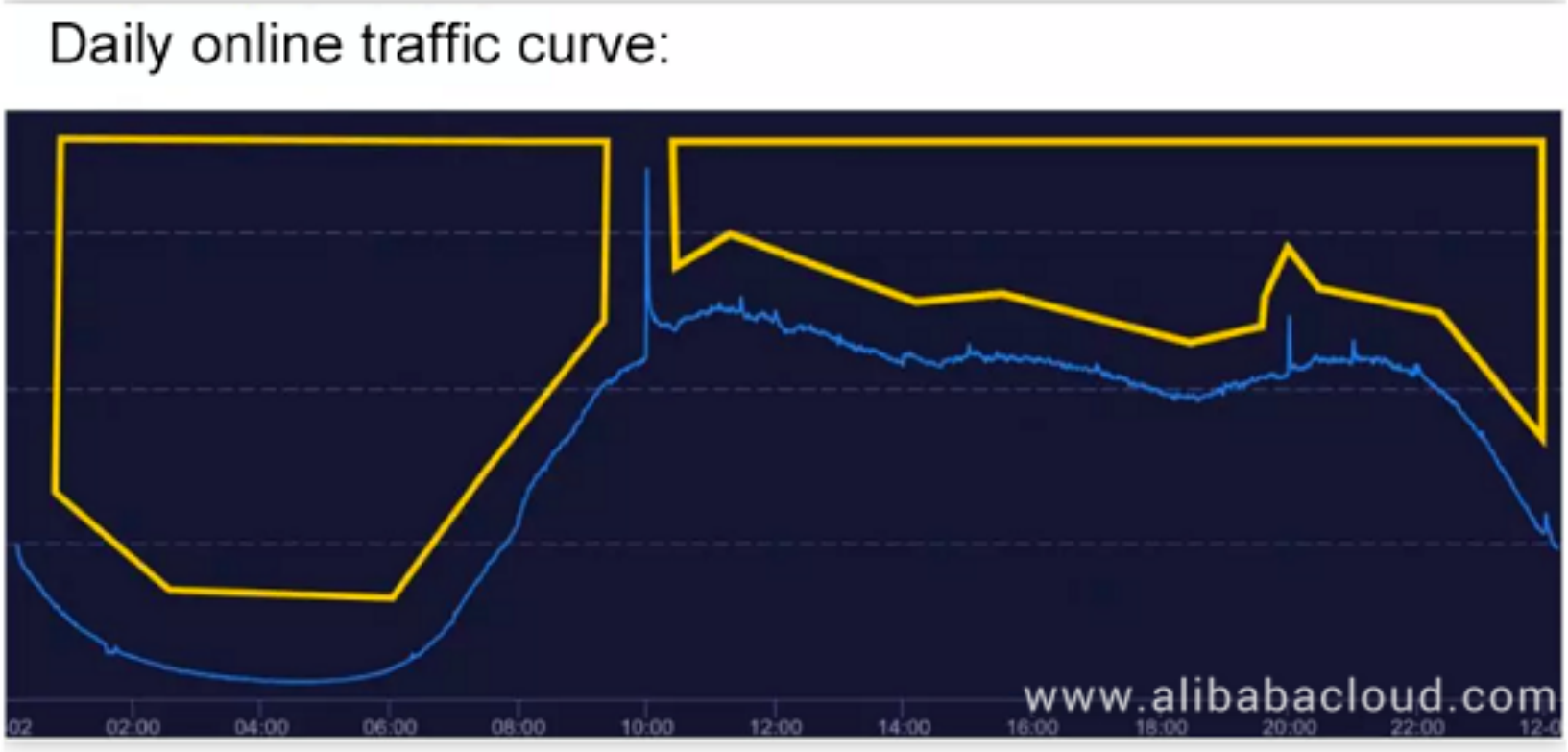

Online service domain
Offline service domain
DB
Transaction
Middleware
Advertisement
Real-time
Online level-2 resource scheduling
Co-location management
Co-location monitoring
Comprehensive decision making
ODPS
ADS
Galaxy
Blink
Online resource scheduler (Sigma)
Unified resource management (level-0)
Resource status
Resource isolation
Offline resource scheduler (Fuxi)
Sigma Agent
Unified agent (level-0 agent)
Fuxi Agent
Online resource pool
Co-location resource pool
Offline resource pool
Infrastructure
IDC
Network
Physical machine
IP address management
Network change
Hardware procurement
Monitoring and alarm
Troubleshooting

Sigma


With colocation 40%
Without colocation 10%
30%
90
80
70
60
50
40
30
20
10
0
07:00
08:00
09:00
10:00
11:00
cpu 平均值
cpu 最大值
cpu 最小值
80
70
60
50
40
30
10
0
07:00
08:00
09:00
10:00
11:00
cpu 平均值
cpu 最大值
cpu 最小值

Sigma

Sigma


Daily online traffic curve:
www.alibabacloud.com
02:00
04:00
06:00
08:00
10:00
12:00
14:00
16:00
18:00
20:00
22:00

Sigma

- Where is the Level-0 agent?
  - A. Master
  - B. Node
- What are the key issues of Level-0 data and controller?

Foundation of Internet Platform Development & Operation

Scheduling II

2019-10-29

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Architecture Details

- High-level summary of scheduling objectives
  - Minimize the job queueing time, or more generally, the system response time (queueing + service times)
  - Subject to
    - Priorities among jobs
    - Per-job constraints
    - Failure tolerance
    - Scalability
- Scheduling architectures
  - Monolithic schedulers
  - Statically partitioned schedulers
  - Two-level schedulers
  - Shared-state schedulers (cf. Omega paper)

Monolithic Scheduler

- Single centralized instance
  - Typical of HPC setting
  - Implements all policies in a single code base
  - Applies the same scheduling algorithm to all incoming jobs
- Alternative designs
  - Support multiple code paths for different jobs
  - Each path implements a different scheduling logic
    - Difficult to implement and maintain

Statically Partitioned Scheduler

- Standard "Cloud-computing" approach
  - Underlying assumption: each framework has complete control over a set of resources
  - Depend on statically partitioned, dedicated resources
  - Examples: Hadoop 1.0, Quincy scheduler
- Problems with static partitioning
  - Fragmentation of resources
  - Sub-optimal cluster utilization

Two-level Scheduler

- Obviates the problems of static partitioning
  - Dynamic allocation of resources to concurrent frameworks
  - Use a “logically centralized” coordinator to decide how many resources to grant

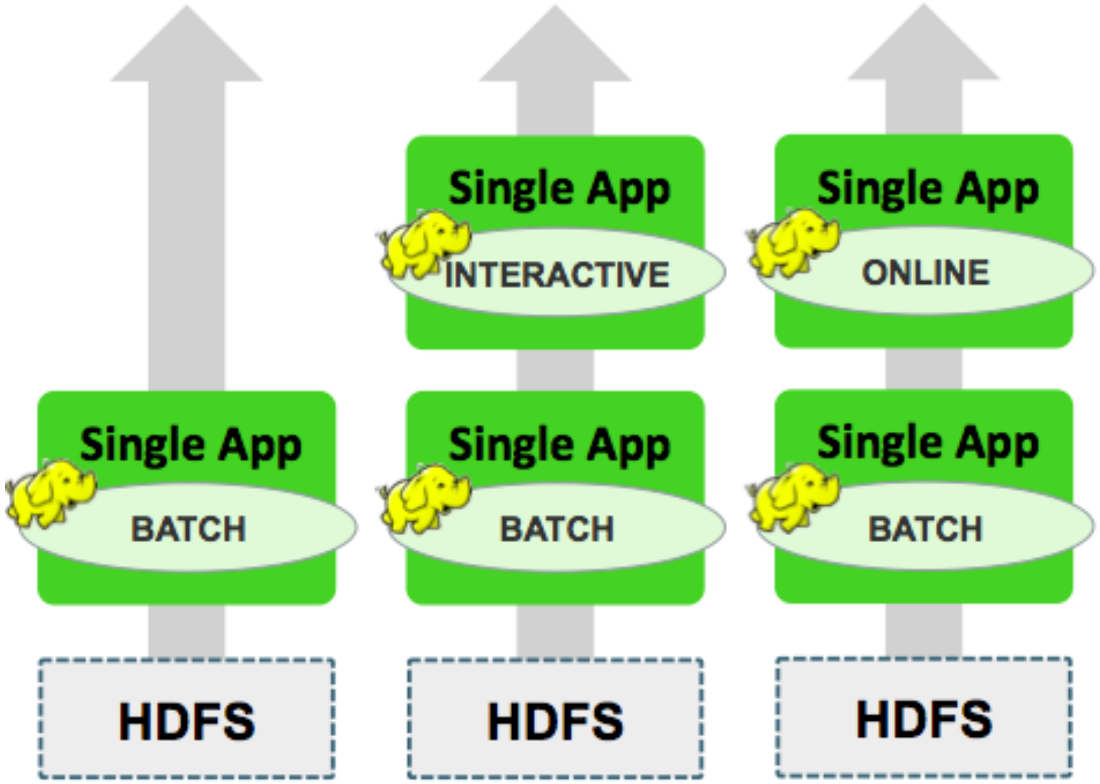
Two-level Scheduler

- A first example: Mesos
  - Centralized resource allocator, with dynamic cluster partitioning
  - Available resources are offered to competing frameworks
  - Avoids interference by exclusive offers
  - Frameworks lock resources by accepting offers
    - Pessimistic concurrency control
  - No global cluster state available to frameworks

Two-level Scheduler

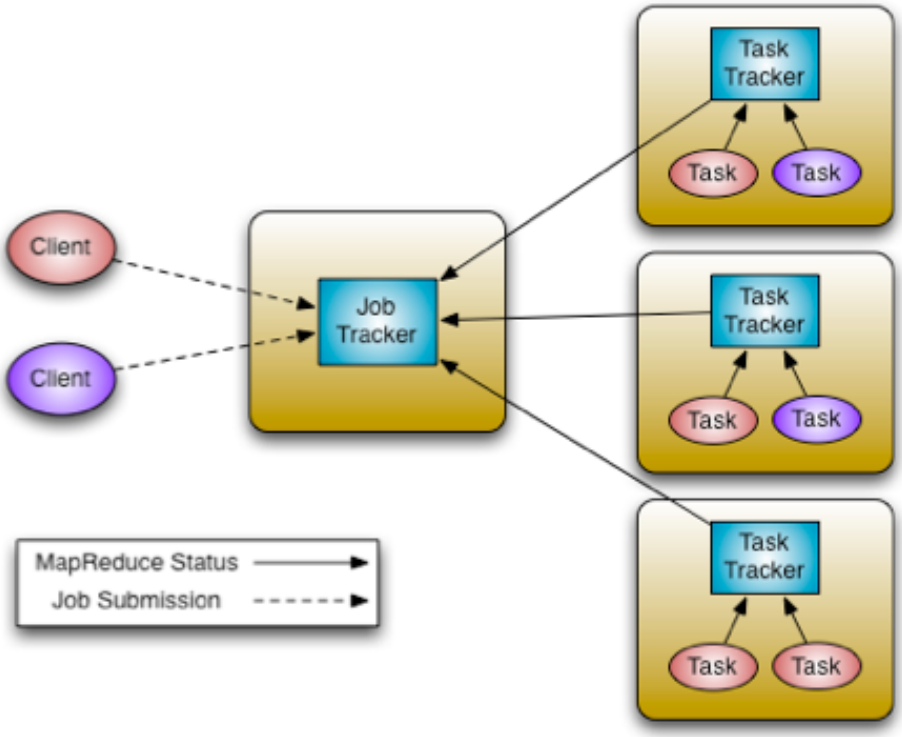
- Another tricky example: YARN
  - Centralized resource allocator (RM), with per-job framework master(AM)
  - AM only provides job management services, not proper scheduling
    - YARN is closer to a monolithic architecture

YARN


Single App
INTERACTIVE
Single App
ONLINE
Single App
BATCH
Single App
BATCH
Single App
BATCH
HDFS
HDFS
HDFS

YARN

- JobTracker
  - Manages cluster resources
  - Performs Job scheduling
  - Performs Task scheduling
- TaskTracker
  - Per machine agent
  - Manages Task execution



YARN

- Only supports MapReduce, no other paradigms
  - Everything needs to be cast to MapReduce
  - Iterative applications are slow
- Scalability issues
  - Max cluster size roughly 4,000 nodes
  - Max concurrent tasks, roughly 40,000 tasks
- Availability
  - System failures destroy running and queued jobs
- Resource utilization
  - Hard, static partitioning of resources in Map or Reduce slots
  - Non-optimal resource utilization

YARN

Single Use System

Batch Apps

HADOOP 1.0

MapReduce
(cluster resource management & data processing)

HDFS
(redundant, reliable storage)

Multi Purpose Platform

Batch, Interactive, Online, Streaming, …

HADOOP 2.0

MapReduce
(data processing)

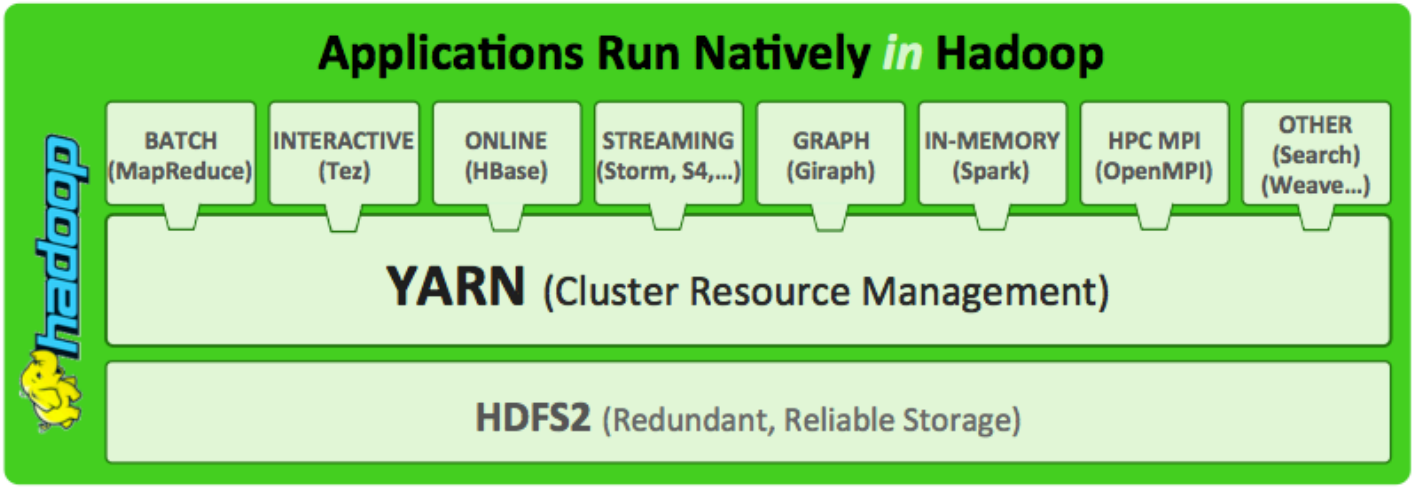
Others

YARN
(cluster resource management)

HDFS2
(redundant, highly-available & reliable storage)

YARN

- Store all data in one place
  - Avoids costly duplication
- Interact with data in multiple ways
  - Not only in batch mode, with the rigid MapReduce model
- More predictable performance
  - Advanced scheduling mechanisms



YARN

- Support for multiple applications
  - Separate generic resource brokering from application logic
  - Define protocols/libraries and provide a framework for custom application development
  - Share same Hadoop Cluster across applications
- Improved cluster utilization
  - Generic resource container model replaces fixed Map/Reduce slots
  - Container allocations based on locality and memory
  - Sharing cluster among multiple application
- Improved scalability
  - Remove complex app logic from resource management
  - State machine, message passing based loosely coupled design
  - Compact scheduling protocol

YARN

- Application Agility
  - Use Protocol Buffers for RPC gives wire compatibility
  - Map Reduce becomes an application in user space
  - Multiple versions of an app can co-exist leading to experimentation
  - Easier upgrade of framework and applications
- A data operating system: shared services
  - Common services included in a pluggable framework
  - Distributed file sharing service
  - Remote data read service
  - Log Aggregation Service