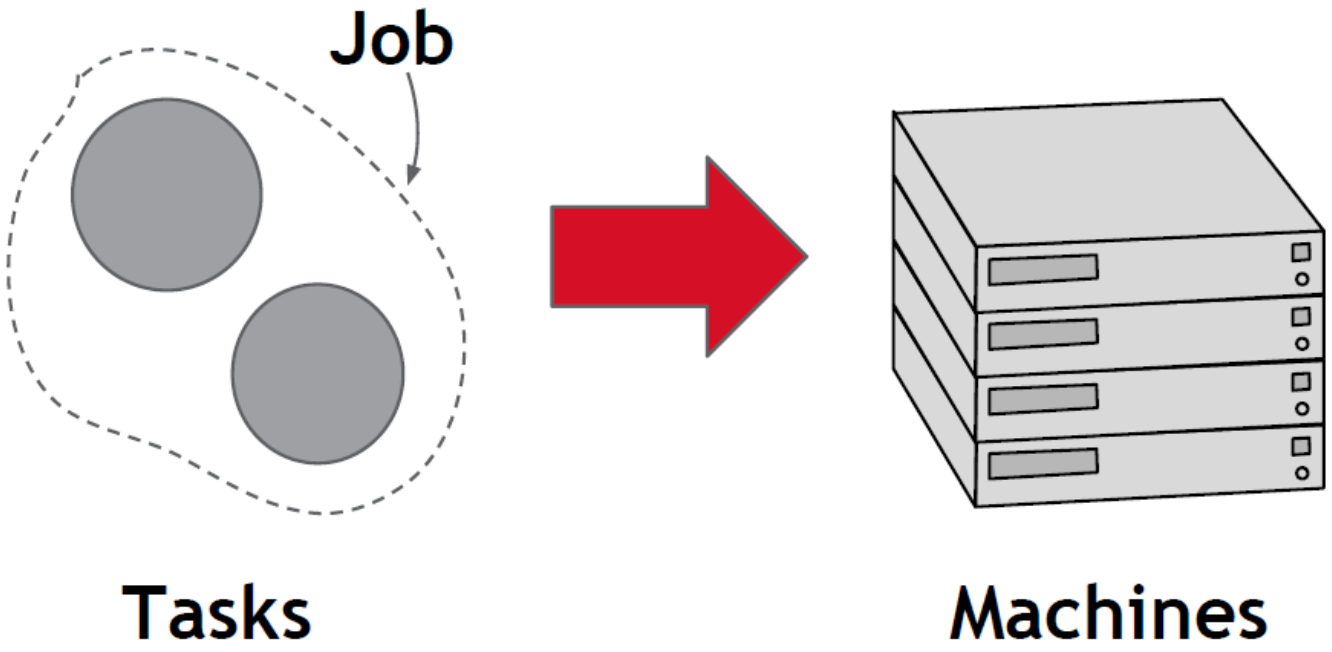
北京航空航天大學
BEIHANG UNIVERSITY

阿里云计算
Alibaba Cloud Computing

Computing at Scale: Resource Scheduling Architectural Evolution and Introduction to Fuxi System

Renyu Yang（杨任宇）
Supervised by Prof. Jie Xu
Ph.D. student@ Beihang University
Research Intern @ Alibaba Cloud Inc.
Member of **CoLAB**(**Co**llaboration of **Leeds**, **Alibaba**, **Beihang**)
2014.8.19 @Tsinghua University

Resource Scheduling Problems



Challenges: Computing at Scale

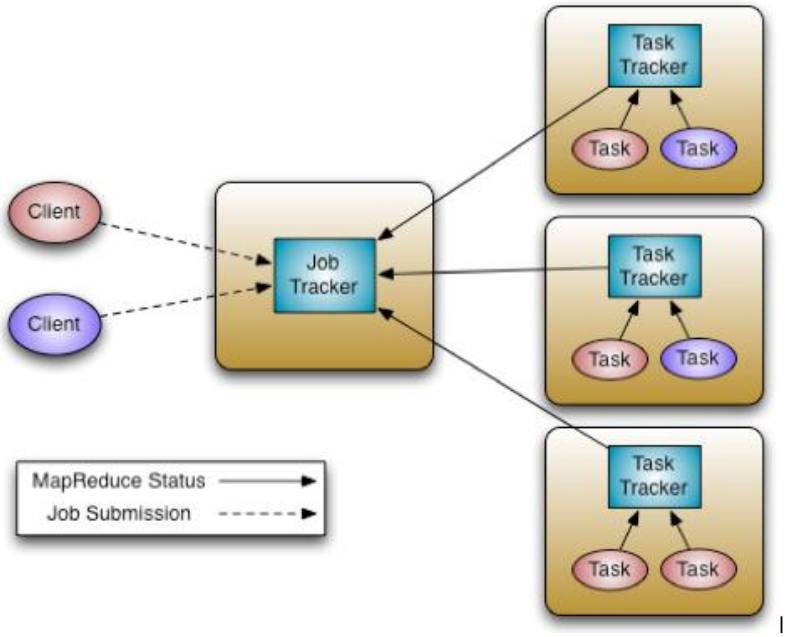
- Perspectives：
  - Performance
  - Scalability
  - Fault-tolerance
  - Resource Utilization

- Case studies:
  - System Architectural Evolution
  - Fuxi system

Resource Management System Overview (non-official)

- **1st Generation(G1)**:
  - Hadoop Map Reduce programming paradigm
- **2nd Generation(G2)**:
  - Mesos
  - Yarn: short for "Yet Another Resource Negotiator"
  - Fuxi: Alibaba Cloud Inc.
- **3rd Generation**？
  - Omega from Google

G1: Hadoop MR classic



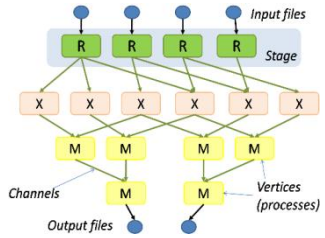
- JobTracker
  - Manages cluster resources
  - Task scheduling
    - how to divide tasks
    - which node, how much resource for execution
- TaskTracker
  - Per-machine agent and daemon
  - Manage each execution individual task

Limitations

- Problems with large scale
  - > 4000 nodes
  - > 40k concurrent tasks
- Only support one one type of computing paradigm (Slots only for Map or Reduce)
- Problems with resource utilization
- Overloaded Job Tracker, Single point of failure!
- Restart is very tricky due to complex states → weak failover mechanism

Increasing Requirement

- Rapid innovation in cluster computing frameworks

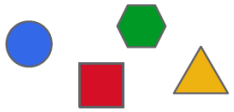


Dryad

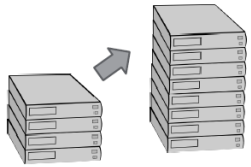
Google
Pregel

Spark

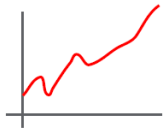
Trend observed !



Diverse workloads



Increasing cluster sizes



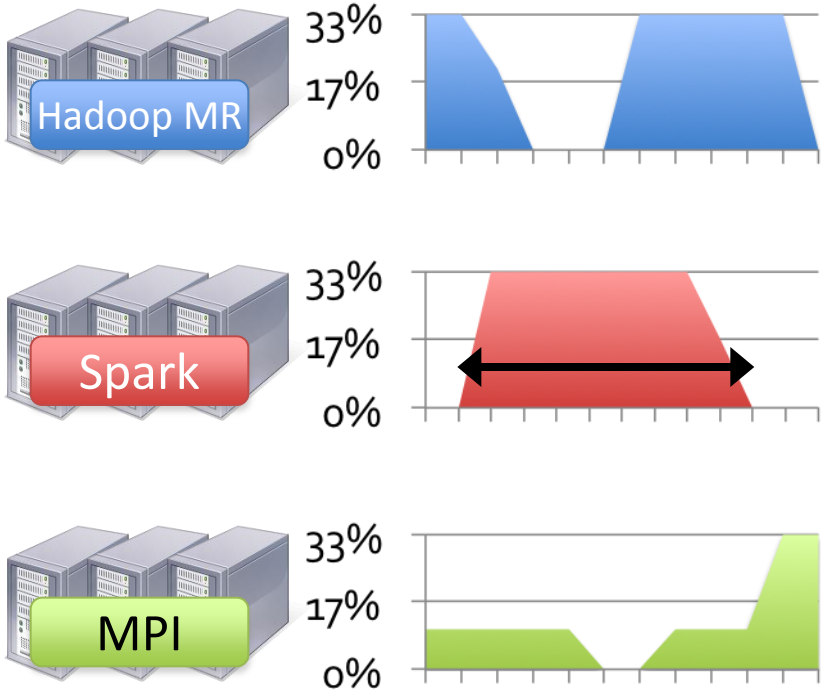
Growing job arrival rates

Users would like to run both existing and new application frameworks on the same physical clusters and at the same time.

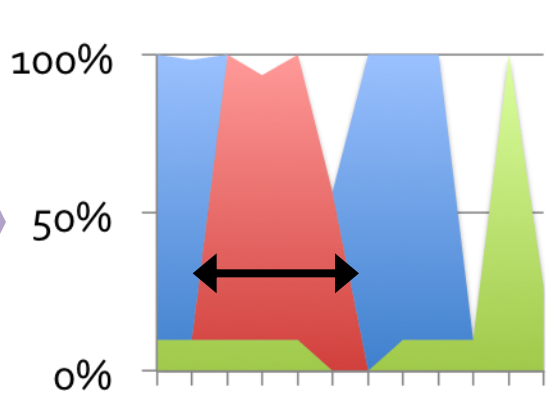
Motivation

No single framework optimal for all applications

Today: static partitioning



dynamic sharing



Shared cluster

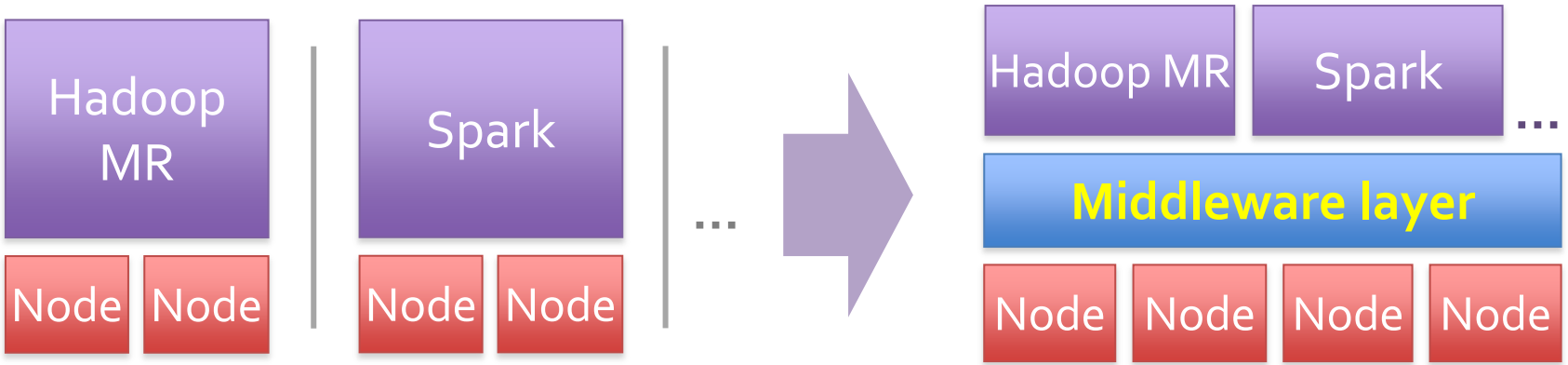
We want to run multiple frameworks in a single cluster

…to *maximize utilization*

…to *share data* between frameworks

2nd Generation(G2) Solutions

- We need a common resource sharing layer over which diverse frameworks can run.



- Goals
  - **High utilization of resources**
  - **Support diverse frameworks**
  - **Better scalability to 10,000's of nodes**
  - **Improved reliability in face of failures**

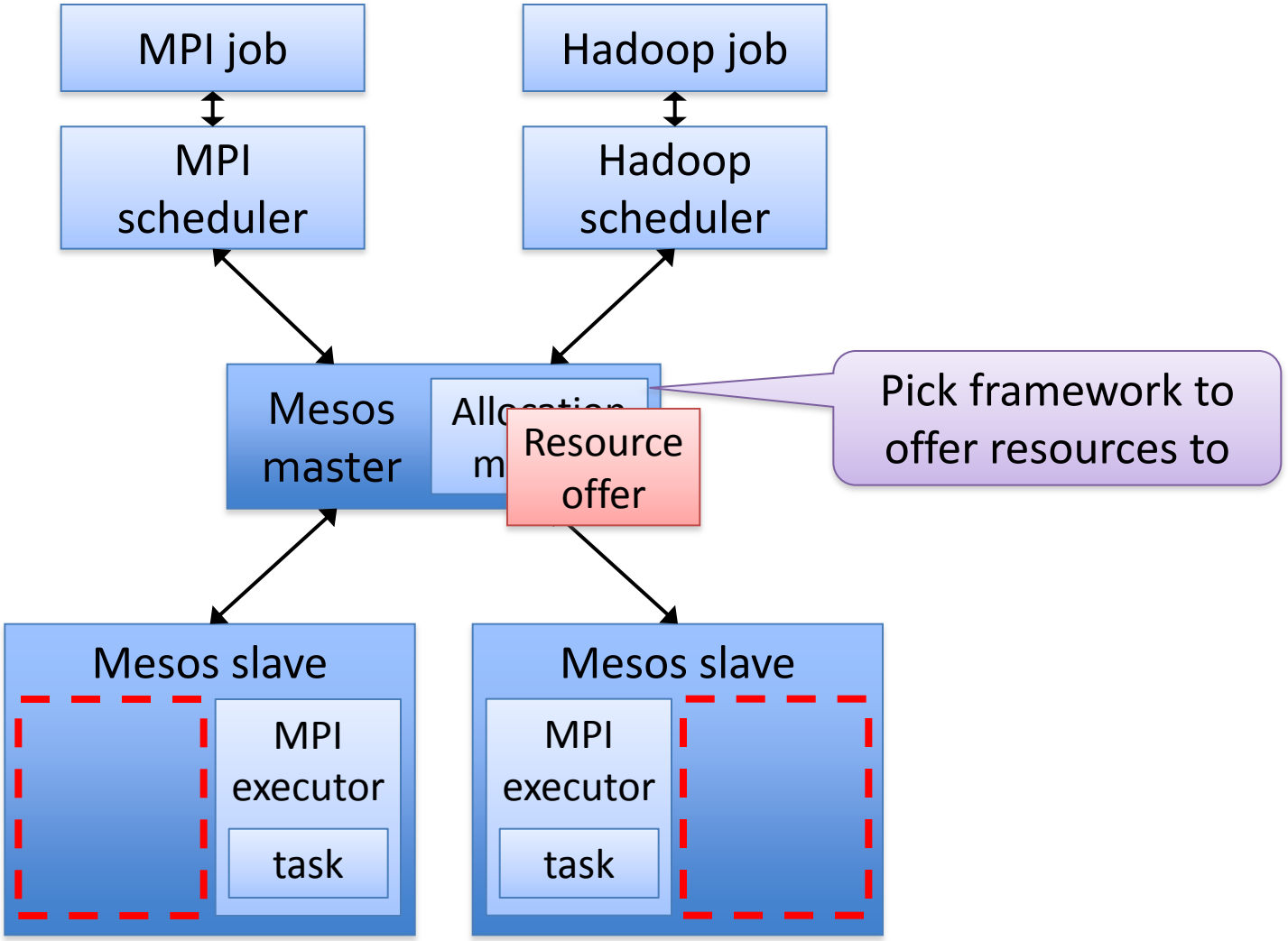
2nd Generation(G2) Core Idea:

- **De-coupling** *the functionalities of JobTracker*:
  - Resource Management
  - Scheduling / Monitoring

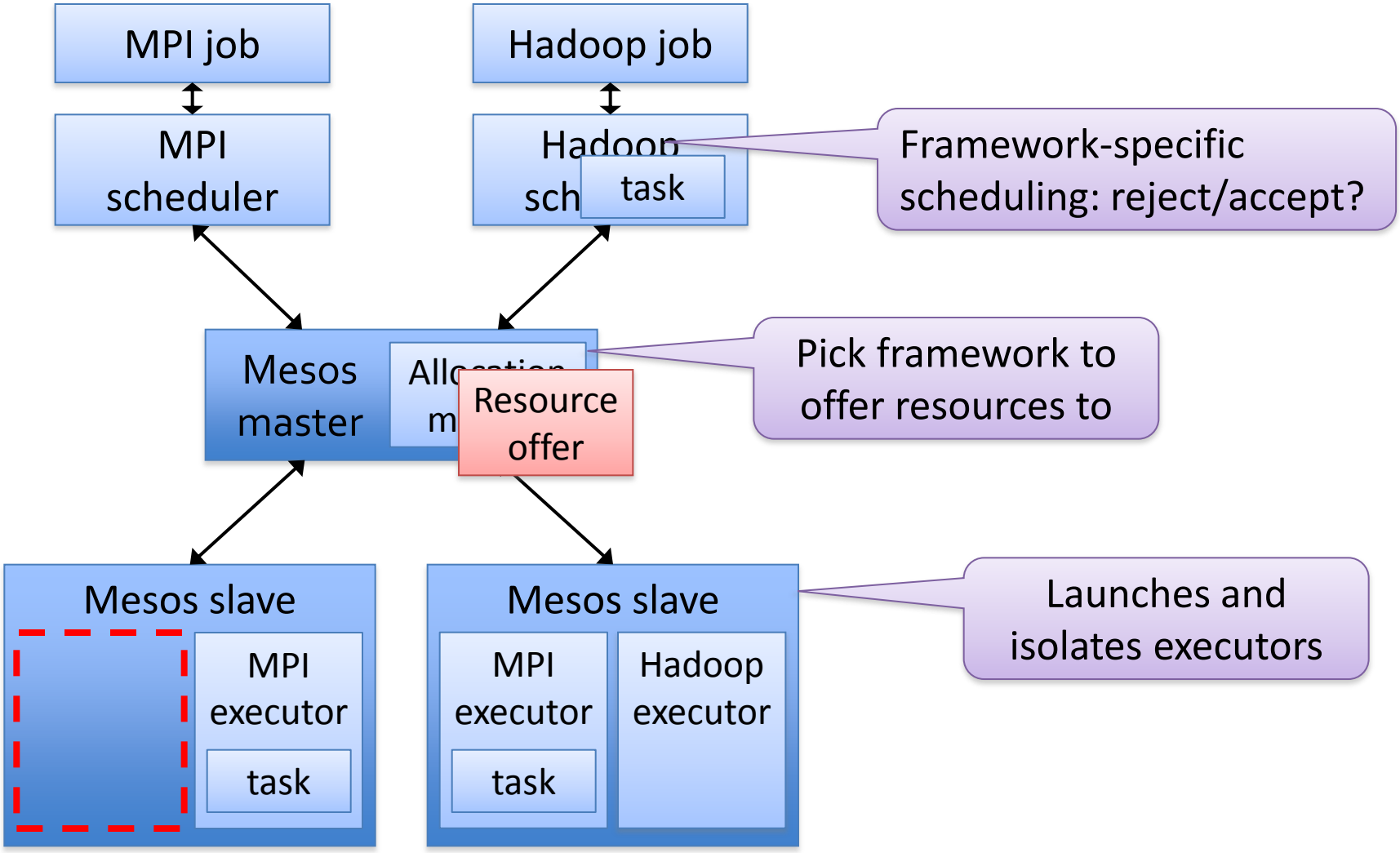
Mesos from Berkeley

- Philosophy: **"offer and accept"** for resource
- **Resource allocation module in Mesos** decides how many resources should be offered to each application framework, based on an organizational policy such as fair sharing (e.g., DRF)
- Frameworks decide which resources to accept and which tasks to run on them.

Mesos Architecture



Mesos Architecture



Limitations

- Passive offer-based policy
  - Only accept or reject what is on offer but cannot specify any request
  - Severed order of each framework depends on the offering order
  - Risk of long-time resource starving
  - Can not support resource preemption

G2++: **Next Generation MRv2**

- De-coupling *JobTracker* into：
  - Resource Management
  - Scheduling / Monitoring

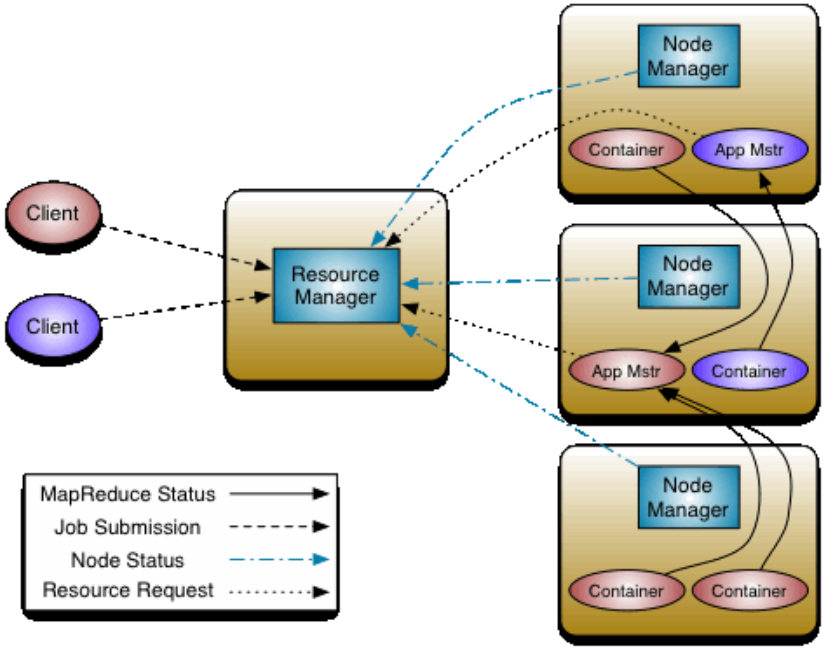
- Following a request-based and active approach that improves scalability, resource utilization, fault tolerance.

- Providing slots for jobs other than Map / Reduce

YARN

- ***JobTracker* is de-coupled into**
  - Global Resource Manager - Cluster resource management
  - Application Master - Job scheduling and monitoring (one per application). The Application Master negotiates resource containers from the Scheduler, tracking their status and monitoring for progress. Application Master itself runs as a normal container.
- ***TaskTracker* is simplified into**
  - NodeManager (NM) - A new per-node slave that is responsible for launching the applications' containers, monitoring their resource usage (CPU, memory, disk, network, etc) and reporting back to the Resource Manager.
- YARN maintains compatibility with existing MapReduce applications and users.

Yarn's Architecture and Workflow



1) Client -> Resource Manager

Submit App Master

2) Resource Manager -> Node Manager

Start App Master

3) Application Master -> Resource Manager

Request containers

4) Resource Manager -> Application Master

response allocated containers

5) Application Master -> Node Manager

Assign resources to tasks(assignment)

Start tasks in containers(start Container-> stop container)

6) Node Manager -> Resource Manager

report running and *terminated container, trigger new round of scheduling.*

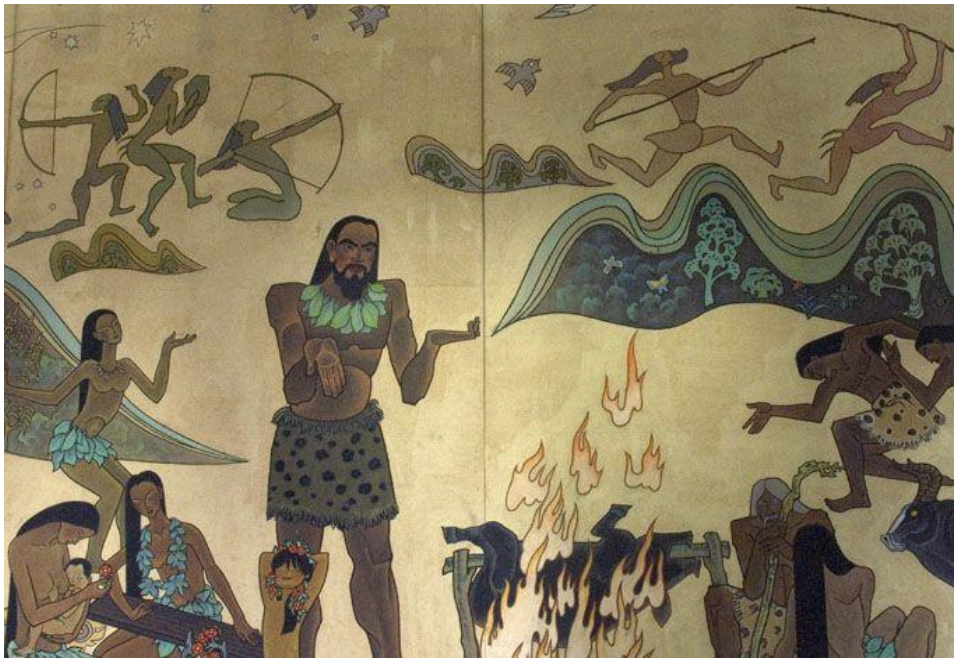
Limitations

- **Scheduling dimensions**
  - (only CPU and memory) are limited and not easy to extend.
- **Scalability issues**
  - Resource assignment at the granularity of a task instance
  - The allocated resource to each container has to be reclaimed by NM once it terminates even if the application has more ready tasks to run.
  - RM has to conduct additional rounds of rescheduling.
  - At most 4k-nodes[1].

Limitations

- **Failover mechanism** is extremely poor to support larger scale:
  - *RM:*
    - Non-transparent Resource Manager Failover
    - merely recover and restore its own states.
    - **AMs cannot survive RM restart**.
  - *NM&AM:*
    - It **uses mandatory termination(mark running container to ” killed”)**
    - **simply re-dos** failed/killed applications.
    - leading to substantial wastes and overheads.
- Possible reason: Yarn directly inherits from Hadoop1.0 in open-source community

Fuxi (伏羲)



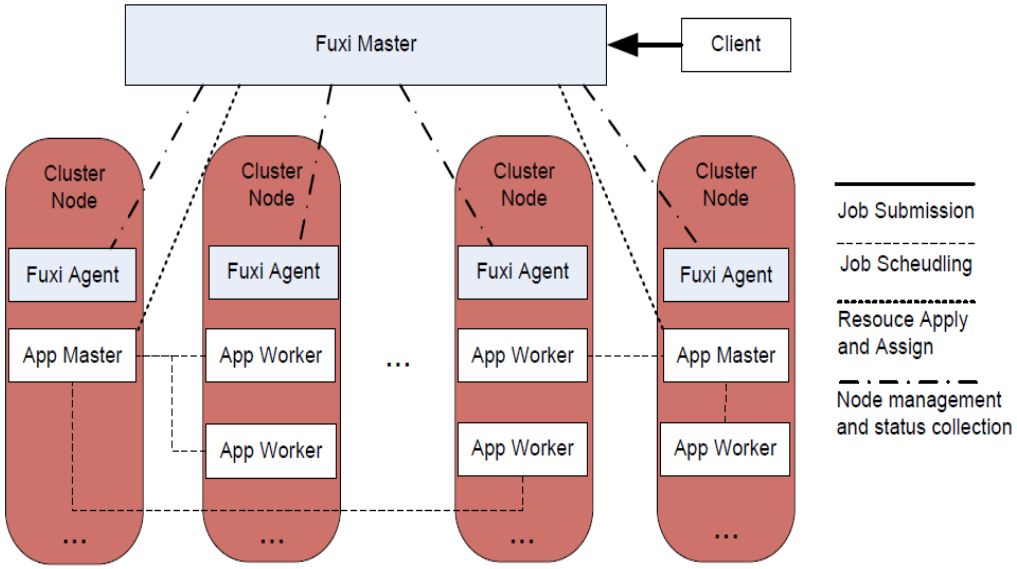
- 伏羲是我国古籍中记载的最早的王。伏羲聪慧过人，他根据天地万物的变化，发明创造了八卦。八卦可以推演出许多事物的变化，预卜事物的发展。

The name of our system, Fuxi, was derived from the first of the Three Sovereigns of ancient China and the inventor of the square and compass, trigram, Tai-Chi principles and the calendar, thereby being metaphorized into a powerful dominator in our system.

Fuxi System

阿里云计算 Alibaba Cloud Computing

- The Fuxi architecture has similarities to YARN



FuxiMaster acts as resource manager

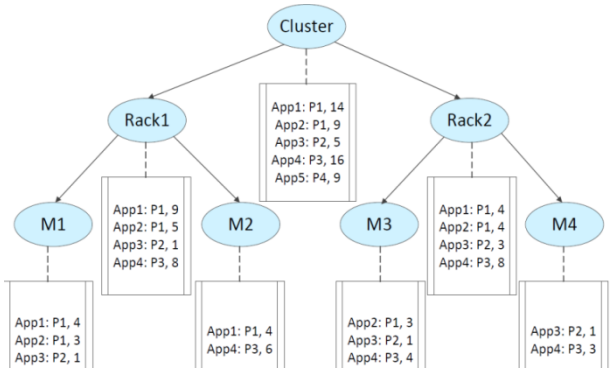
FuxiAgent acts as Node manager

App master serves the same function

- Focus on two challenging problems:
  - ***Scalability (+ Efficiency):***
  - How to support 5k or more nodes but avoiding message floods?
  - How to achieve hundreds of thousands of requests per second?
  - ***Fault-Tolerance (+ Efficiency):***
  - How to provide transparent failover mechanism?

Improved Scalability

- **Incremental scheduling and communication**
  - Resource *request* is only sent once until the application master *release* the resources.
    - Scheduling tree and multi-level waiting queue with priority identified.
    - New round of scheduling is triggered only when resources release.
  - An incremental request will be sent only when the resource demands are dynamically adjusted.
    - Reducing frequency of message passing
    - Improving the whole cluster utilization
- **Multi-dimension resource allocation**
  - CPU, mem, other virtual resources



Tree-based scheduling example (multi-level waiting queues)

Advanced Fault Tolerance

- **User-transparent failover mechanisms**
  - **Resource Manager:**
    - Refill ***hard states*** *(*meta-data, configuration file etc.) from light-weight checkpoint with no impact to running applications.
    - collect ***soft states*** from App Masters, Node Managers in run-time.
  - **Application Master:** can also performs failover to recover the finished and running workers by all task instances' snapshot.
  - **Node Manager:** rebuild the complete states with full granted resource and worker list from each app master.

Advanced Fault Tolerance

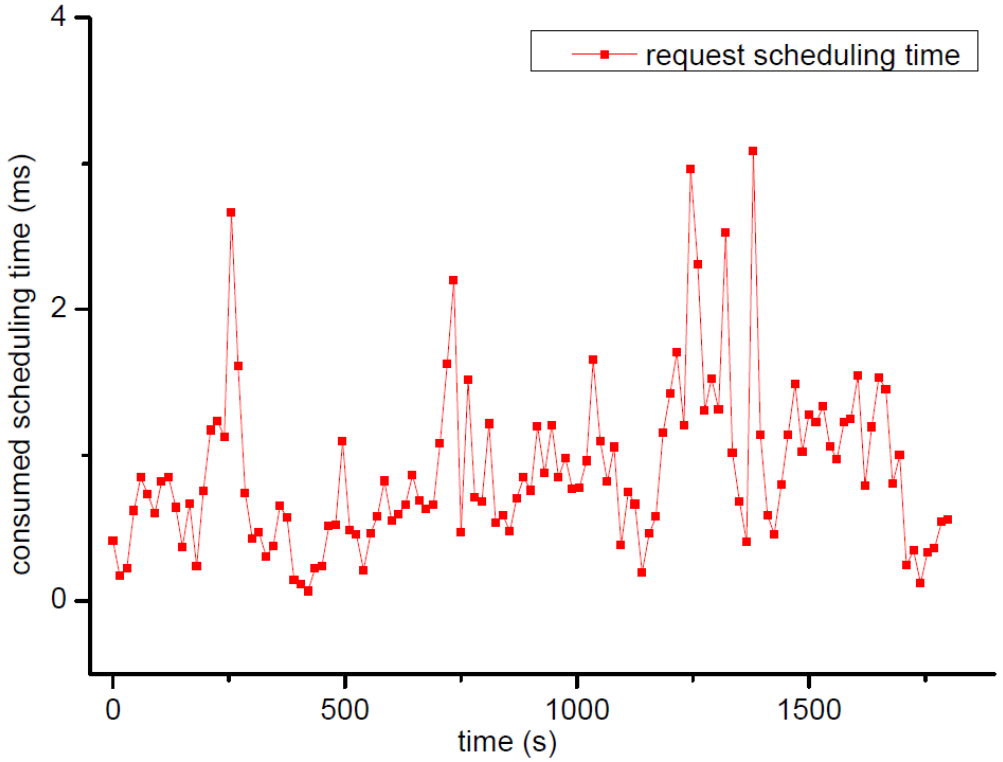
- **"Bad" (frequently failed) node detection and multilevel blacklist.**
  - **Cluster-level**
    - Heartbeat threshold control
    - Application-level information collection
    - Plug-in service to aggregate OS-level information
  - **Task-level**
    - If one task instance is reported failed on one machine, the machine will be added into the instance's blacklist.
  - **Job-level**
    - When marked as "bad" by a certain number of instances, the machine will not be used by this job.
- **Reduce the negative impact of faulty nodes on scheduling and decrease the probability of failures.**

Experimental Evaluation

- **Environment Setup**
  - **5,000 servers** with each machine consisting of 2.20GHz 6 cores Xeon E5-2430, 96GB memory and 12*2T disk.

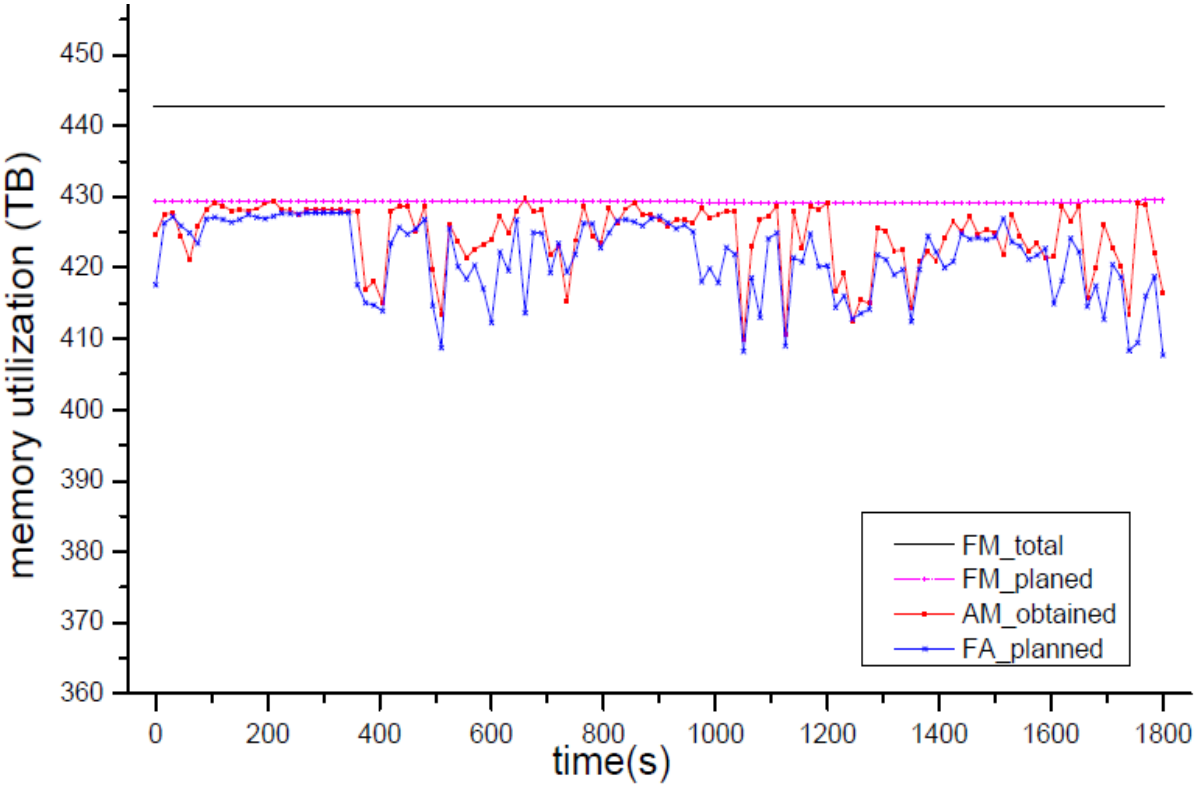
- **Objectives:**
  - **Scheduling performance** when 1,000 jobs are submitted simultaneously to a cluster.
  - **Sort benchmarks** are utilized to illustrate the performance and capability of task execution in Fuxi.
  - Several **fault-injection** experiments are carried out

Job Scheduling Time in Fuxi



The average scheduling time takes merely **0.88 ms** and the peak time consumption for scheduling is no more than **3 ms**, demonstrating that the system is rapid enough to reclaim the allocated resource as well as respond to incoming requests in a timely manner.

Scheduled Resource in Fuxi



- Up to 97.1% of resources will be initially utilized by the scheduler.
- Gaps amongst these curves indicate overheads of processing requests by Fuxi master (In fact no more than 5%).
- The actual resource consumption by FuxiAgents is often less than requested due to user's overestimation.

Sort Benchmark

Table 2: GraySort Indi Result Comparison

Provenance	Configurations	GraySort Indi Result
Fuxi (2013)	5000 nodes (2 2.20GHz 6cores Xeon E5-2430, 96 GB memory,12x2TB disks)	100TB in 2538 seconds(2.364TB/min)
Yahoo!Inc. (2012)	2100 nodes (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory,12x3TB disks)	102.5 TB in 4,328 seconds(1.42TB/min)
UCSD (2011)	52 nodes (2 Quadcore processors,24 GBmemory, 16x500GB disks) Cisco Nexus 5096 switch	100 TB in 6,395 seconds(0.938TB/min)
UCSD&VUT (2010)	47 nodes (2 Quadcore processors,24 GB memory, 16x500GB disks) Cisco Nexus 5020 switch	100 TB in 10,318 seconds(0.582TB/min)
KIT (2009)	195 nodes x (2 Quadcore processors,16 GB memory,4x250GB disks)288-port InfiniBand 4xDDR switch	100 TB in 10,628 seconds(0.564TB/min)

End to end execution time in Fuxi is 2,538s which is equivalent to 2.364TB/minute, achieving 66.5% improvement, as compared with the most advanced Yahoo!'s Hadoop implementation

See more results (e.g. fault injection experiments) in our recent paper

Publications

- Collaborated Paper will appear as the coming VLDB 2014 full paper

***Fuxi*: a Fault-Tolerant Resource Management and Job Scheduling System at Internet Scale**

Zhuo Zhang*, Chao Li*, Yangyu Tao*, Renyu Yang†*, Hong Tang*, Jie Xu†§
Alibaba Cloud Computing Inc.* Beihang University† University of Leeds§
{zhuo.zhang, li.chao, yangyu.taoyy, hongtang}@alibaba-inc.com
yangry@act.buaa.edu.cn j.xu@leeds.ac.uk

ABSTRACT

Scalability and fault-tolerance are two fundamental challenges for all distributed computing at Internet scale. Despite many recent advances from both academia and industry, these two problems are still far from settled. In this paper, we present Fuxi, a resource management and job scheduling system that is capable of handling the kind of workload at Alibaba where hundreds of terabytes of data are generated and analyzed everyday to help optimize the company's business operations and user experiences. We employ several novel techniques to enable Fuxi to perform efficient scheduling of hundreds of thousands of concurrent tasks over large clusters with thousands of nodes: 1) an incremental resource management protocol that supports multi-dimensional resource allocation and data locality; 2) user-transparent failure recovery where failures of any Fuxi components will not impact the execution of user jobs; and 3) an effective detection mechanism and a multi-level blacklisting scheme that prevents them from affecting job execution. Our evaluation results demonstrate that 95% and 91% scheduled CPU/memory utilization can be fulfilled under synthetic workloads, and Fuxi is capable of achieving 2.36TB/minute throughput in GraySort. Additionally, the same Fuxi job only experiences approximately 16% slowdown under a 5% fault-injection rate. The slowdown only grows to 20% when we double the fault-injection rate to 10%. Fuxi has been deployed in our production environment since 2009, and it now manages hundreds of thousands of server nodes.

1. INTRODUCTION

We are now officially living in the *Big Data* era. According to a study by Harvard Business Review in 2012, 2.5 exabytes of data are generated everyday and the speed of data generation doubles every 40 months [13]. To keep up with the pace of data generation, data processing has also been progressively migrating from traditional database-based approaches to distributed systems that scale out much easily. In recent years, many systems have been proposed from both academia and industry to support distributed data processing with commodity server clusters, such as Mesos [11], Yarn [18] and Omega [16]. However, two major challenges remain unsettled in these systems when faced with the challenges of managing resources for systems at Internet scale:

1) ***Scalability***: Resource scheduling can be simply considered as the process of matching demand (requests to allocate resources to run processes) with supply (available resources of cluster nodes). So the complexity of resource management is directly affected by the number of concurrent tasks and the number of server nodes in a cluster. Furthermore, other factors also impact the complexity, including supporting resource allocation over multiple dimensions (such as CPU, memory, and local storage), fairness and quota constraints across competing applications; and scheduling tasks close to data. A naive approach of delegating everything decision to a single master node (as in Hadoop 1.0) would be severely limited by the capability of the master. On the other hand, a fully-decentralized solution would be hard to design to satisfy scheduling constraints that depends on fast-changing global states without high synchronization cost (such as quota management). Both Mesos and Yarn attempt to deal with these issues in slightly different ways. Mesos adopts a multiple-level resource offering framework. However, Mesos master offers free resources in turn among frameworks, the waiting time for each framework to acquire desired resources highly depends upon the resource offering order and other frameworks' scheduling efficiency. Yarn's architecture decouples resource management and programming paradigms. However, the decoupling is only for the separation of code logic between resource management and MapReduce job execution, so that other programming paradigms can be accommodated in Yarn. Nevertheless, it does not reduce the complexity of resource management, and still inherits the linear resource model as in Hadoop 1.0.

In both cases, states are exchanged with periodic messages and the interval configuration is another intricate challenge. A long period could reduce communication overhead but would also hurt utilization when applications wait for resource assignment. On the other hand, frequent adjustments would improve response to demand/supply changes (and thus improve resource utilization); however, it would also aggravate the message flooding phenomenon. In fact, Yahoo! reported that they did not run Hadoop clusters bigger than 4,000 nodes [18] and Yarn had yet to demonstrate its scale limit.

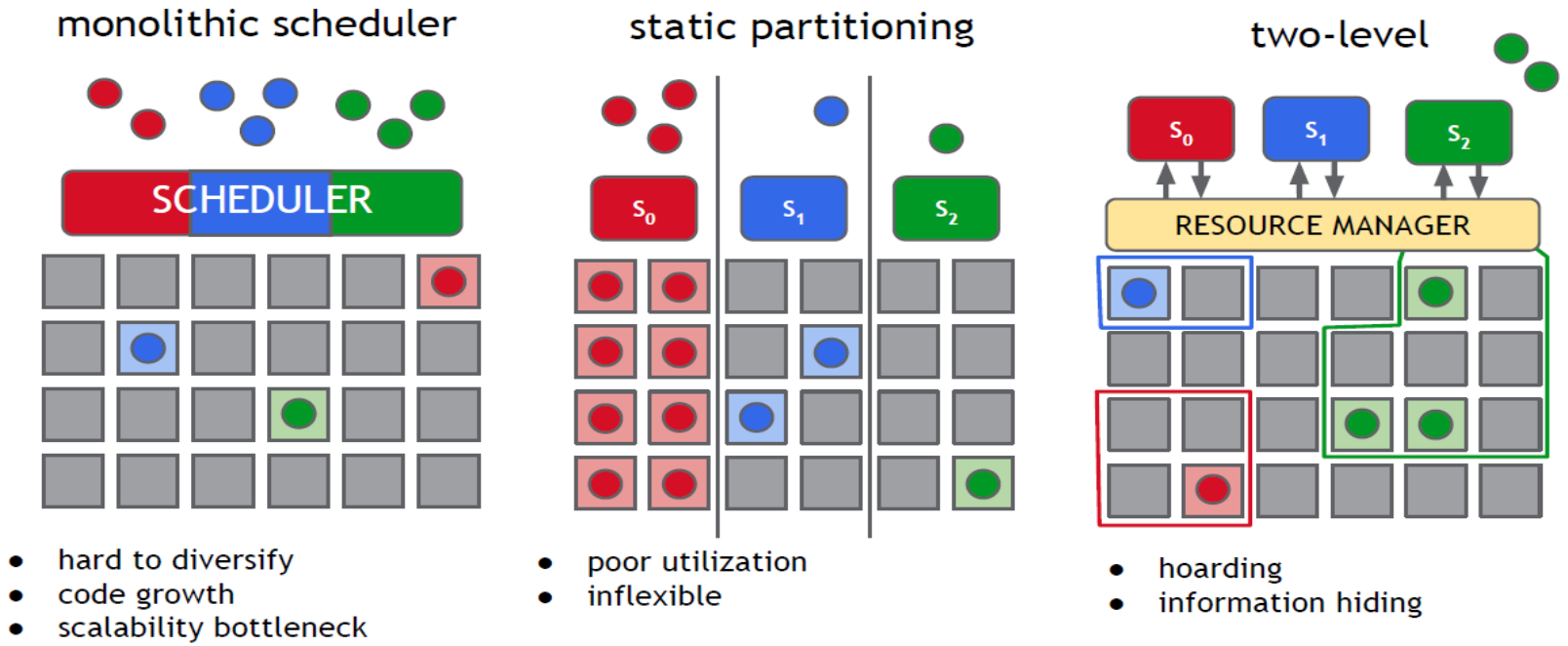
2) ***Fault-tolerance***: With increasing scale of a cluster,

Proceedings of the VLDB Endowment, Vol. 7, No. 13

[1] Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, and Jie Xu. ***Fuxi: a Fault-Tolerant Resource Management and Job Scheduling System at Internet Scale***, The 40th International Conference on Very Large Data Base (VLDB 2014), Hangzhou, China, 2014 (to appear)

[2] Renyu Yang, Peter Garraghan, Zhuo Zhang, Jie Xu, Hua Cai etc. ***Root-Cause Monitoring, Analysis and Failure Recovery for Large-Scale Cloud Datacenters***, *in IEEE Transaction on Reliability (In preparation)*

Google's view of multiple framework mixture



- hard to diversify
- code growth
- scalability bottleneck

- poor utilization
- inflexible

- hoarding
- information hiding

Mesos/Yarn/Fuxi

- Conservative resource-visibility and locking algorithms limit both flexibility and parallelism.
- Hard to place difficult-to-schedule "picky" jobs or to make decisions that require access to the state of the entire cluster.

Third Generation? – Google's view

- Google proposes ***shared state*** by using lock-free optimistic concurrency control.
  - Lock-free optimistic concurrency control over the shared states, collected timely from different nodes.
  - Each application framework holds the same resources view and competes for the same piece of resource with a central **coordination component** for arbitration.

Potential Limitations in Google's Omega

- More efficient? Let's wait and see…
- It is hard to enforce global properties such as capacity/fairness/deadlines.
- It may be tough for heterogeneous frameworks outside Google when sharing the same cluster.
- Current simulation-based evaluation needs more real-life experience and practice.

Reference

[1] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In Proc. SoCC, page 5. ACM, 2013.
[2] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In Proc. EuroSys, pages 351-364. ACM, 2013
[3] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In Proc. NSDI, Usenix, 2011.
[4] J. Dean and S. Ghemawat. Mapreduce: simplifed data processing on large clusters. Communications of the ACM, 51(1):107-113, 2008.
[5] Apache Hadoop. http://hadoop.apache.org/
[6] http://www.pdl.cmu.edu/SDI/2013/slides/wilkes-sdi.pdf
[7] Zhuo Zhang, Chao Li, Yangyu Tao, Renyu Yang, Hong Tang, Jie Xu. Fuxi: a Fault Tolerant Resource Management and Job Scheduling System at Internet Scale (to appear in Proceeding of VLDB 2014 vol.7) , Sept 2014.
[8] SLS: http://hadoop.apache.org/docs/r2.3.0/hadoop-sls/SchedulerLoadSimulator.html

Thank you for your attention!

Renyu Yang, Ph.D. student
School of Computing
Beihang University, Beijing, China
Homepage: *http://act.buaa.edu.cn/yangrenyu*
Email: *yangry@act.buaa.edu.cn*

CoLAB