Cake
Enabling High-level SLOs on Shared Storage Systems

AMPLab – UC Berkeley
Introduction

- **Cake**
  - *Coordinated* (multiresource) 2-level scheduling system for shared storage systems
  - Enforces *SLO* requirements of the clients.
Introduction

- **Cake**
  - **Coordinated 2-level scheduling system for shared storage systems**

2 classes of datacenter applications

- **Front End Web Server**
  - User Facing
  - Latency Sensitive

- **Internal Batch Analytics**
  - Throughput-Oriented

Different workloads, requirements
Service Level Objective

Performance Metric (Ex: 99th percentile latency) of Requirement (Ex: 100 ms) for Type of Request (Ex: get request)
Service Level Objective (Examples)

- **Latency SLO:**
  - 99th percentile latency of 100ms for get requests

- **Throughput SLO:**
  - 100 8KB scan requests per second

- 90% of calls to the helpdesk should be answered in less than 20 seconds…
Introduction

- Cake
  - Coordinated 2-level scheduling system for shared storage systems

- Schedules different types of requests received by clients.
- Grants access to a hardware resource – cpu, disk.
- **REMEMBER**: It is a user-level scheduling system (on top of OS scheduler)
Introduction

- Cake
  - Coordinated 2-level scheduling system for shared storage systems

- Provides scheduling at multiple resources — cpu, disk.
- Scheduling at different resources is coordinated. Why?
Introduction

- Cake
  - Coordinated 2-level scheduling system for shared storage systems

- Both classes of applications share a distributed storage system – HBase, HDFS, Cassandra etc.
Motivation

- Separate storage systems for both the classes of applications.
  - Hard to multiplex front-end (web server) storage and backend (analytics) storage.
  - Different workloads and requirements for both these classes.
  - We don’t want to violate SLOs of web-clients.
Motivation

Front End

Backend
Motivation

Front End

Backend

Users → Data → Analysis

Anirudh Ravula | UIUC | CS 525 | 4/11/13
Observation:
Separate storage systems for front-end and back-end applications

Problem:
Hard to multiplex latency sensitive applications and throughput-oriented applications.
Why? – inherent latency v/s throughput tradeoff in rotational media storage

Solution:
Have separate queue each for latency sensitive applications requests and throughput-oriented applications requests
Schedule them by giving higher priority for latency sensitive applications so that SLOs are met
Cake Objectives

- Combine both the workloads
  - consolidate separate front-end and batch storage clusters
- Meet front-end latency requirements
- Then maximize batch throughput
Cake Benefits

Consolidated

- Lower cost of provisioning
- Better cluster utilization
- Improved performance.
- Also meets SLOs
Recap

- **Cake**
  - *Coordinated multiresource scheduler*
  - Can *consolidate storage systems* for different types of workloads (latency sensitive vs throughput-oriented)
  - Also meets SLO requirements
System Design

Client

GET

RESPONSE

Process

HBase

lookup

READ

READ DATA

HDFS
System Design

Client

\[\begin{align*}
\text{HBase} & \quad \text{RPC Handlers} \\
\text{HDFS} & \quad \text{CPU Usage} \\
\text{HBase} & \quad \text{RPC Handlers} \\
\text{HDFS} & \quad \text{Disk Usage}
\end{align*}\]
System Design

- Responsible for scheduling at single resource (disk, cpu etc)
- Built into RPC layer of each software component
System Design

- **2nd–level-Scheduler**
- **1st–level-Scheduler**

**Client**

- RPC Handlers
- CPU Usage
- Disk Usage

**HBase**

- Performs Metrics from 1st–level-schedulers.
- Coordinates the resource allocation at 1st–level-schedulers.
- Enforces SLOs
System Design

- Add scheduling points within the system
- Dynamically adjust allocations to meet SLO requirements.
- Evaluated on HBase/HDFS system.
Schedulers: Overview

- **First-Level Schedulers**
  - Give control over underlying hardware resource: cpu, disk
  - Implemented at different layers of software stack (HBase, HDFS)

- **Second-Level Scheduler**
  - Coordinates first-level schedulers to decide allocations
  - Enforces SLO requirements
System Design
First-Level Schedulers

- Provide **effective control** over the underlying hardware – cpu, disk etc.
- **Coordinate** with second-level scheduler
Effective scheduler requirements

- Differentiated Scheduling
- Split large requests
- Control number of outstanding requests
FIFO Scheduling Scheme

Problems
- Unfairness with a single FIFO queue
- Front-end requests block behind batch requests

Solution:
- Separate queues for both classes of applications
Schedule based on allocations set by the 2nd-level scheduler

Allocations:
- Proportional share allocation
- Reservations

Problems:
- Large requests might tie up resources
- Requests can be non-preemptible

Solution: split large requests
System Design
First-Level Schedulers – Splitting large requests

- Split large requests into multiple chunks
- Only wait for a chunk than an entire large request
- Tradeoff:
  - Lower latency but lesser throughput
- For the experiments performed, 64KB chunk size was found to be optimal
System Design
First-Level Schedulers – Limiting outstanding requests

- A device can only handle a certain number of concurrent executions
- Need to make sure the thread pool size at HBase/HDFS is optimal
  - Not overwhelming the device
  - Not underloading the device
TCP congestion control technique: **AIMD** (additive increase multiplicative decrease)

- Periodically determine the device latency
- If device underloaded, additively increase # of threads
- If device overloaded, multiplicatively decrease # of threads
- Claim: converges in general case
System Design
Second-Level Scheduler

- Decides allocation at first-level schedulers
  - Collects **performance** and **queue occupancy** metrics from first-level schedulers.
  - Every **interval** (10 secs) – uses these metrics to decide scheduling allocations at first-level schedule
  - Enforces front-end client’s SLOs.
System Design
Second-Level Scheduler

- **2 phases in allocation**
  - **SLO Compliance-based**
    - Adjusts allocations at HDFS.
  - **Queue Occupancy-based**
    - Adjusts HBase allocation based on queue occupancy at HDFS and HBase
System Design
Second-Level Scheduler : SLO Compliance-Based Phase

- Adjusts HDFS allocations
- Disk is the bottleneck in storage workloads
- Clients performance $< SLO$
  - Increase allocation when performance $< SLO$
- However, if client’s performance is very good
  - Decrease allocation
Disk bottleneck workloads. But HBase can be bottleneck too (processing of get requests at HBase can be expensive)

HBase can throttle HDFS

Balance HBase/HDFS allocation

Queue occupancy metric:
- % of time a client’s requests is waiting in the queue at a resource

Increase allocation when more queuing at HBase
Decrease allocation when more queuing at HDFS
Evaluation

- Several challenging consolidated workload scenarios
- Yahoo! Cloud Serving Benchmark (YCSB) clients to generate simulated front-end and batch load.
- Front-end clients – configured to make single-row requests (8KB data)
- Batch MapReduce clients configured to make 500-row scans (4MB data)
- C1.xlarge EC2 instances
Evaluation : Diurnal Workload Scenario

- Traces obtained from a web-server workload of an “industrial partner”
- Front-end running web serving workload
- Batch client running at max throughput

Goals
- Evaluate ability to adapt to dynamic workload patterns
- Evaluate latency v/s throughput trade-off
Evaluation : Diurnal Workload Scenario

Front-end throughput according to diurnal pattern

Time (seconds)

Throughput (queries/s)

3x difference
Details/Observations

- **3 experiments** with different front-end latency SLOs (100ms, 150ms, 200ms)
- Observe that we **miss** the 100ms SLO slightly. The 99th percentile latency for this experiment is at 105ms.
- Throughput **increases** as latency requirements relax. Traditional in rotational storage media

<table>
<thead>
<tr>
<th>Front-end 99th SLO (in ms)</th>
<th>% of requests meeting latency requirements</th>
<th>Batch Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>98.77</td>
<td>24.6 queries/s</td>
</tr>
<tr>
<td>150</td>
<td>99.72</td>
<td>41.2 queries/s</td>
</tr>
<tr>
<td>200</td>
<td>99.88</td>
<td>41.2 queries/s</td>
</tr>
</tbody>
</table>
Evaluation : Diurnal Workload Scenario

Second-level scheduler actions at HBase and HDFS for 100ms SLO

SLO compliance-based algorithm

Queue occupancy-based algorithm
Evaluation : Spike Workload Scenario

- Goal – evaluate the ability of the system to deal with sudden traffic spikes

The spike workload considered for the experiment

![Spike Workload Diagram](image-url)
Observations

- **3 experiments** with different front-end latency SLOs (100ms, 150ms, 200ms)
- Observe that we miss the 100ms SLO slightly. The 99th percentile latency for this experiment is at 107ms.
- Throughput increases as latency requirements relax. Traditional in rotational storage media
- 200 ms SLO achieves higher throughput than diurnal case.

<table>
<thead>
<tr>
<th>Front-end latency SLO (ms)</th>
<th>Batch Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>22.9 queries/s</td>
</tr>
<tr>
<td>150</td>
<td>38.4 queries/s</td>
</tr>
<tr>
<td>200</td>
<td>45 queries/s</td>
</tr>
</tbody>
</table>
Evaluation : Convergence Time

Workload :
Front-End workload : 100 qps +
Unthrottled batch client
Latency SLO placed: 150 ms 99th percentile
Scheduler Interval : 10s

Observations
• 40 secs to fall below SLO
• 150 secs to stabilize.

Convergence for dynamic workloads like diurnal?
Evaluation : Analytics

- 20-node EC2 cluster
- Front-end YCSB client running diurnal pattern
- Batch MapReduce scanning over 386GB data

Goals:
- Quantifying benefits of consolidating separate front-end and backend storage clusters
- Evaluate analytics time and provisioning cost
- Evaluate SLO compliance
Evaluation: Analytics

Performance Gains:
1.7x speedup +
50% provisioning cost

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Time</th>
<th>Speedup</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconsolidated</td>
<td>1722s</td>
<td>1.0x</td>
<td>40</td>
</tr>
<tr>
<td>Consolidated</td>
<td>1030s</td>
<td>1.7x</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Front-end 99th SLO</th>
<th>% Requests Meeting SLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>100ms</td>
<td>99.95%</td>
</tr>
</tbody>
</table>
Evaluation: Summary

- Can adapt to **changing** workload pattern
- Can adapt to **spike** workload pattern
- Can adjust SLOs to give **control** over latency v/s throughput tradeoff

**Performance:**
- **1.7 x speedup** in batch/analytics jobs
- **50%** provisioning cost
Discussion

- Convergence times on dynamic workload? Is Cake guaranteed to converge?
- SLOs on throughput?
- No experiments on write workloads. Some SLOs were violated for read requests. They could be more severe for writes.
- Extensibility to new storage abstractions?
  - Not possible to implement 1st level scheduling criteria at all layers. Chunking not applied at HBase.
- Future Work: SLO admission control, Application-level SLOs, use of SSDs, parameter tuning, multiple SLOs.
Dominant Resource Fairness: Fair Allocation of Multiple Resource Types

Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, Ion Stoica *

NSDI 2011

* University of California, Berkeley, CA

Presented By

Md Tanvir Al Amin
University of Illinois at Urbana-Champaign, IL
Resource Demands in a Datacenter

One month (Oct 2010) trace of CPU and Memory demands for a 2000-node Hadoop cluster at Facebook.
Schedulers in Practice

- Slot based schedulers
  - Allocate resources at the granularity of **slots**
  - Slot is a **fixed fraction** of a node
  - **Agnostic** of user demand **heterogeneity**

- Example
  - Quincy (Dryad)
  - Hadoop’s Fair Scheduler

- Outcome?
  - Underutilization
  - Thrashing
  - Users can game the system
It really happens!

- Users game the system (Not strategy-proof)
  - Yahoo! Hadoop cluster allocated more slots for Reduce
  - User launched all his jobs as long reduce phases
  - A “Big” search company provided dedicated machines for jobs that had high utilization
  - Users inserted artificial infinite loops

- Underutilization and Over-utilization
  - CDF of demand to slot ratio in the Facebook example.
    - ratio < 1 : Underutilizing
    - ratio > 1 : Over-utilizing
It really happens!

- **Users game the system (Not strategy-proof)**
  - Yahoo! Hadoop cluster allocated more slots for Reduce
  - User launched all his jobs as long reduce phases
  - A “Big” search company provided dedicated machines for jobs that had high utilization
  - Users inserted artificial infinite loops

- **Underutilization and Over-utilization**
  - CDF of demand to slot ratio in the Facebook example.
  - ratio < 1 : Underutilizing
  - ratio > 1 : Over-utilizing

60% tasks need more CPU
95% had at least double of what really required
Problem Definition

• How to
  • Fairly Share
    • Multiple type of resources
  • Among
    • Different Users
• When users have
  • Heterogeneous demands
Fairness Policy

• **Sharing incentive**
  - Each user should get \textit{at-least} \( \frac{1}{n} \) fraction of the cluster

• **Strategy-proof**
  - One cannot ‘cheat’ by lying about demand

• **Envy-free**
  - User should not prefer allocation of the other

• **Pareto-efficiency**
  - Cannot increase allocation of a user without the expense of another
Fairness Policy

- **Single resource fairness**
  - Reduce to max-min fairness in single resource scenario

- **Bottleneck fairness**
  - Reduce to max-min on bottleneck resource if it is the only dominant resource

- **Population monotonicity**
  - A user leaves ➔ Other users allocation do not decrease

- **Resource monotonicity**
  - Resource added ➔ No users allocation decrease
Single Resource Fairness

- **Max-Min Fairness strategy**
  - Allocate chunks in the order of increasing demand
  - Nobody gets more than what it asks
  - All unsatisfied demands get an equal share
Single Resource Fairness

- **Max-Min Fairness strategy**
  - Allocate chunks in the order of increasing demand
  - Nobody gets more than what it asks
  - All unsatisfied demands get an equal share

*Animation adapted from https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx
Single Resource Fairness

- **Max-Min Fairness strategy**
  - Allocate chunks in the order of increasing demand
  - Nobody gets more than what it asks
  - All unsatisfied demands get an equal share

*Animation adapted from [https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx](https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx)
Single Resource Fairness

- **Max-Min Fairness strategy**
  - Allocate chunks in the order of increasing demand
  - Nobody gets more than what it asks
  - All unsatisfied demands get an equal share

*Animation adapted from https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx*
Single Resource Fairness

- **Max-Min Fairness strategy**
  - Allocate chunks in the order of increasing demand
  - Nobody gets more than what it asks
  - All unsatisfied demands get an equal share

---

Round Robin, TCP, Fair Queueing, etc. all try to approximate Max-Min Fairness

Maximizes the **Minimum** share of the unsatisfied ones

*Animation adapted from [https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx](https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx)*
Single Resource Fairness

- Max-Min Fairness strategy
  - Allocate chunks in the order of increasing demand
  - Nobody gets more than what it asks
  - All unsatisfied demands get an equal share

Only “reasonable” mechanism with Sharing incentive and Strategy-proof properties

Round Robin, TCP, Fair Queueing, etc. all try to approximate Max-Min Fairness

Maximizes the Minimum share of the unsatisfied ones

*Animation adapted from https://theone.ece.cmu.edu/drupal/sites/default/files/papers/Dominant%20Resource%20Fairness.pptx
Max-Min Fairness

- Multiple Resource
  - 2 resources: CPUs & memory
  - User 1 wants <1 CPU, 4 GB> per task
  - User 2 wants <3 CPU, 1 GB> per task
- What is a fair allocation?
- Users have tasks according to a demand vector
- Not needed in practice, can simply measure actual consumption
- Assume divisible resources

A Natural Policy: Asset Fairness

- **Asset Fairness**

  - Equalize each user’s *sum of resource shares*

- Cluster with 70 CPUs, 70 GB RAM

  - $U_1$ needs $\langle 2 \text{ CPU, 2 GB RAM} \rangle$ per task
  - $U_2$ needs $\langle 1 \text{ CPU, 2 GB RAM} \rangle$ per task

- Asset fairness yields

  - $U_1$: 15 tasks: 30 CPUs, 30 GB ($\Sigma=60$)
  - $U_2$: 20 tasks: 20 CPUs, 40 GB ($\Sigma=60$)

---

A Natural Policy: Asset Fairness

- **Asset Fairness**
  - Equalize each user’s *sum of resource shares*

- **Cluster with 70 CPUs, 70 GB RAM**

  Problem
  User 1 has < 50% of both CPUs and RAM
  Better off in a separate cluster with 50% of the resources

- **Asset fairness yields**
  - $U_1$: 15 tasks: 30 CPUs, 30 GB ($\Sigma=60$)
  - $U_2$: 20 tasks: 20 CPUs, 40 GB ($\Sigma=60$)

Dominant Resource Fairness

- A user’s **dominant resource** is the resource she has the biggest share of

- Example:
  
  Total resources: \(<10 \text{ CPU}, 4 \text{ GB}>\)

  User 1’s allocation: \(<2 \text{ CPU}, 1 \text{ GB}>\)

  Dominant resource is memory as \(\frac{1}{4} > \frac{2}{10} (\frac{1}{5})\)

- A user’s **dominant share** is the fraction of the dominant resource she is allocated

- User 1’s dominant share is 25% (\(\frac{1}{4}\))
Dominant Resource Fairness

Example:

Total resources: <9 CPU, 18 GB>
User 1 demand: <1 CPU, 4 GB> dominant res: mem
User 2 demand: <3 CPU, 1 GB> dominant res: CPU

\[
\begin{align*}
\text{max}(x, y) & \\
\text{s.t.} & \\
x + 3y & \leq 9 \\
4x + y & \leq 18 \\
\frac{2x}{9} & = \frac{y}{3} \\
\therefore x & = 3, y = 2
\end{align*}
\]
Dominant Resource Fairness

Example:

Total resources: <9 CPU, 18 GB>
User 1 demand: <1 CPU, 4 GB> dominant res: mem
User 2 demand: <3 CPU, 1 GB> dominant res: CPU

\[
\begin{align*}
\text{max}(x, y) & \quad \text{x tasks from User 1, y tasks from User 2} \\
\text{s.t.} & \\
x + 3y & \leq 9 \quad \text{CPU Constraint} \\
4x + y & \leq 18 \quad \text{Memory Constraint} \\
\frac{2x}{9} & = \frac{y}{3} \quad \text{DRF Equalize}
\end{align*}
\]

User 1: <3 CPU, 12 GB>, User 2: <6 CPU, 2 GB>
Dominant Resource Fairness

Example:

Total resources: <9 CPU, 18 GB>
User 1 demand: <1 CPU, 4 GB> dominant res: mem
User 2 demand: <3 CPU, 1 GB> dominant res: CPU
Online DRF Scheduler

Whenever there are available resources and tasks to run:

*Schedule a task to the user with smallest dominant share*

- Easy computation
  - $O(\log n)$ time per decision using binary heaps
- How to determine demand vectors?
Alternative: CEEI

• Approach
  • Set *prices* for each good
  • Let users buy what they want

• How do we determine the right prices for different goods?
  • Let the market determine the prices

• **Competitive Equilibrium from Equal Incomes (CEEI)**
  • Give each user \( \frac{1}{n} \) of every resource
  • Let users trade in a perfectly competitive market

• Not strategy-proof!
\[
\max (x, y) \\
\text{s.t.} \\
x + 3y \leq 9 \\
4x + y \leq 18 \\
\therefore x = \frac{45}{11}, y = \frac{18}{11}
\]
max\((x,y)\) Product of Nash Products

\[\begin{align*}
\text{st} & \\
x + 3y & \leq 9 \quad \text{CPU Constraint} \\
4x + y & \leq 18 \quad \text{Memory Constraint}
\end{align*}\]

\[
\therefore x = \frac{45}{11}, y = \frac{18}{11}
\]

User 1: <4.1 CPU, 16.4 GB>,
User 2: <4.9 CPU, 1.6 GB>
DRF vs CEEI

- User 1: <1 CPU, 4 GB>  User 2: <3 CPU, 1 GB>

- DRF more fair, CEEI better utilization

DRF vs CEEI

- User 1: <1 CPU, 4 GB>  User 2: <3 CPU, 1 GB>

- DRF more fair, CEEI better utilization

User 1: <1 CPU, 4 GB>  User 2: <3 CPU, 2 GB>

User 2 increased her share of both CPU and memory

DRF vs CEEI

- User 1: <1 CPU, 4 GB>  User 2: <3 CPU, 1 GB>

- DRF more fair, CEEI better utilization

User 1: <1 CPU, 4 GB>  User 2: <3 CPU, 2 GB>

User 2 increased her share of both CPU and memory

---

DRF vs Asset Fairness vs CEEI

- Resources <1000 CPUs, 1000 GB>
- 2 users A: <2 CPU, 3 GB> and B: <5 CPU, 1 GB>
<table>
<thead>
<tr>
<th>Property</th>
<th>Asset Fairness</th>
<th>CEEI</th>
<th>DRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharing Incentive</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Strategy-Proofness</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Envy-freeness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pareto efficiency</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Single Resource Fairness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bottleneck Fairness</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Population Monotonicity</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Resource Monotonicity</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Evaluation

- Micro-experiments
  - 48 node Mesos cluster on EC2
  - Extra large instances with 4 CPU cores and 15 GB of RAM
  - Two jobs, one CPU intensive, one memory intensive
    - Compare DRF with current Hadoop scheduler
  - Macro-benchmark through simulations
    - Simulate Facebook trace with DRF and current Hadoop scheduler
Micro Experiments

Job 1’s Share

Job 2’s share

Dominant Share
Hadoop Fair Scheduler Experiments

• Hadoop Fair Scheduler/capacity/Quincy
  • Each machine consists of k slots (e.g. k=2~6)
  • Run at most one task per slot
    • apply max-min fairness to slot-count
Micro Experiments

Number of large jobs completed

Number of small jobs completed
Micro Experiments

Number of large jobs completed

Number of small jobs completed
Micro Experiments

Average response time for large jobs

Average response time for small jobs
Micro Experiments

Average response time for large jobs

- DRF: 65
- 3 slots: 69
- 4 slots: 72
- 5 slots: 123
- 6 slots: 196
- CPU-fair: 173

Average response time for small jobs

- DRF: 25
- 3 slots: 61
- 4 slots: 39
- 5 slots: 35
- 6 slots: 56
- CPU-fair: 25

Thrashing
Macro Benchmarks

Average reduction of the completion times for different job sizes for a trace from a Facebook Hadoop cluster.
Utilization

CPU and Memory utilization for DRF and slot fairness for a trace from Facebook Hadoop Cluster
Discussions

• Fair sharing vs. Meeting Deadlines? (Indy)
  • Is the job throughput only concern?

• What is the cloud specific requirement behind this research?
  • Why don’t we require / apply this fairness in a single machine OS?
  • Is multi-tenancy of a cloud the only reason for this scheduling?

• Only Memory and CPU! IO, Network, Disk is absent from experiments?
  • Hadoop stores the intermediate results in persistence storage

• They are considering exclusive resources. What about shared resources in the Datacenter? How to share the network?
  • Multi-Resource Fair Queueing for Packet Processing: SIGCOMM 2012
  • Schedule multiple resources in a Middleboxes (IDS, VPN, Firewall, Wan Optimizer etc)

• What about the task placement constraints?
• Integrate DRF with Hadoop?
• Leaves some resource unutilized
• Adding more resources to the system may decrease the allocations for existing users.
  • Proved in the paper that satisfying everything is not possible