When Computation Meets Communication: The Case for Scheduling Resources in the Cloud

Prof. Bo Li
CSE @ HKUST
NOSSDAV, June 23, 2017
Cloud is Everywhere
Cloud is Everywhere

- Cloud Storage: Dropbox
- Service Migration: Netflix (AWS)
- Recommendation Systems
  - Videos you may like
  - Inspired by your shopping trends
- Online Games
- Virtual / Augmented Realities.
Cloud: Massive Scale

- Facebook [GigaOM, 2012]
  - 30K in 2009 -> 60K in 2010 -> 180K in 2012
- Microsoft [DC knowledge]
  - 1M, 2013
- AWS EC2 [Randy Bias]
- Google [DC knowledge]
Datacenter: inside
Server Racks

Photo credit: Google
How do we program the cloud?
MapReduce: a Programming Model for Typical Big Data Problems
Typical Big Data Problems
Typical Big Data Problems

Video Analytics:

- What movies would a Netflix user like to watch?
Typical Big Data Problems

Video Analytics:

› What movies would a Netflix user like to watch?

Log Analysis:

› How many warnings were logged last week?
  › [Warning] 18:34, 06/02/2017: Low Memory
Typical Big Data Problems

Video Analytics:

› What movies would a Netflix user like to watch?

Log Analysis:

› How many warnings were logged last week?
  › [Warning] 18:34, 06/02/2017: Low Memory

Web Mining:

› How many tweets mentioned the word “terrorism” yesterday?
Common Theme in Big Data
Common Theme in Big Data

- A large volume of data that does not fit into one machine
Common Theme in Big Data

- A large volume of data that does not fit into one machine

- Information of interest needs to be extracted, aggregated, and possibly passed to the next iteration
Common Theme in Big Data

- A large volume of data that does not fit into one machine
- Information of interest needs to be extracted, aggregated, and possibly passed to the next iteration

Parallelization is needed!
Divide-and-Conquer

“Work”

map₁  →  worker  →  reduce₁  ↓  “Result”

map₂  →  worker  →  reduce₂

map₃  →  worker  →  reduce₃

Partition

Combine
MapReduce: the Programming Model
MapReduce: the Programming Model

1. Scan through a large number of records
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
3. Shuffle and sort intermediate results
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
3. Shuffle and sort intermediate results
4. Aggregate intermediate results
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
3. Shuffle and sort intermediate results
4. Aggregate intermediate results
5. Repeat 1-4 or generate final output
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each

3. Shuffle and sort intermediate results
4. Aggregate intermediate results
5. Repeat 1-4 or generate final output
MapReduce: the Programming Model

1. Scan through a large number of records
2. Extract something of interest from each
3. Shuffle and sort intermediate results
4. Aggregate intermediate results
5. Repeat 1-4 or generate final output
MapReduce Runtime

hadoop Spark
A MapReduce job consists of $M$ map tasks shuffling intermediate results to $N$ reduce tasks.
Where to Run those Tasks?

How should the system assign tasks to machines to orchestrate *computation* and *communication*?
The Scheduling Problem

John Wilkes, QoS issues in Google cluster management, IWQoS 13
The Scheduling Problem

A multi-dimensional Knapsack problem:

Place tasks w/ different demands onto machines w/ different capacities

\[ \text{max } f(x_{ij}) \]  
subject to  
\[ \sum_j x_{ij} \leq 1, \quad x_{ij} \in \{0, 1\} \]  
\[ \sum_i c_i \cdot x_{ij} \leq \text{CPU}_j, \forall \text{ machine } j \]  
\[ \sum_i r_i \cdot x_{ij} \leq \text{RAM}_j, \forall \text{ machine } j \]  
\[ \sum_i n_i \cdot x_{ij} \leq \text{bandwidth}_j, \forall \text{ machine } j \]
Multi-dimensional Knapsack Problem

A 2-dimensional example

Memory (GB)

0

# CPU Cores
Multi-dimensional Knapsack Problem

A 2-dimensional example

Memory (GB) vs. # CPU Cores
Multi-dimensional Knapsack Problem

A 2-dimensional example
Multi-dimensional Knapsack Problem

A 2-dimensional example

Allocated cores and memory
Multi-dimensional Knapsack Problem

A 2-dimensional example

Allocated cores and memory
Task Scheduling is Hard!

Diversified workload:
e.g., network- or computation-intensive

Increasing cluster size:
the problem scale blasts.

Growing job arrival rates:
make decisions within a few milliseconds.
Even Harder

- **Constraints on data locality:**
  Tasks are better placed onto machines w/ local input data

- **App-specified placement constraints:**
  e.g., place tasks on machines w/ Linux-4.9 or above

- **Data dependency:**
  - Downstream tasks cannot start until upstream tasks have completed
Scheduling: Objectives

- Performance
  - Run as many tasks as quickly as possible
- Fairness
  - A datacenter is shared by multiple tenants
- Other objectives
  - Fault tolerance, scalability, energy efficient, etc.
Scheduling: Objectives

- Performance
  - Run as many tasks as quickly as possible
- Fairness
  - A datacenter is shared by multiple tenants
Computation Performance

- Minimize the average task completion time
- Shortest task first: a 2-machine example

pending tasks with different processing times

2 idle machines, serve 1 task at a time

machine 1  machine 2
Computation Performance
Computation Performance

- Shortest task first
Computation Performance

- Shortest task first
  - Optimal for a single machine
Computation Performance

- Shortest task first
  - Optimal for a single machine
  - NP-hard even for two machines
Computation Performance

- Shortest task first
  - Optimal for a single machine
  - NP-hard even for two machines

![Diagram showing computation performance between two machines over time.](diagram.png)
Computation Performance

- Shortest task first
  - Optimal for a single machine
  - NP-hard even for two machines
Computation Performance

- Shortest task first
  - Optimal for a single machine
  - NP-hard even for two machines
Computation Performance

- Shortest task first
  - Optimal for a single machine
  - NP-hard even for two machines
Computation Performance

- Shortest task first
  - Optimal for a single machine
  - NP-hard even for two machines
MapReduce Runtime

Communication barrier: all intermediate results have to be received before next stage of computation
The Coflow abstraction: a collection of flows transmitting intermediate results of a job

The coflow of a 2-by-2 MapReduce job: only when all the four flows of coflow1 finish, will coflow1 finish.

M. Chowdhury et al., Efficient Coflow Scheduling with Varys, SIGCOMM 14
Communication Efficiency
Communication Efficiency

- Minimize the average coflow completion time
Communication Efficiency

- Minimize the average coflow completion time
- Coflow scheduling: prioritize “shortest” coflow
Communication Efficiency

- Minimize the average coflow completion time
- Coflow scheduling: prioritize “shortest” coflow

coflow 1 = (F₁, F₂, F₃, F₄)
Communication Efficiency

- Minimize the average coflow completion time
- Coflow scheduling: prioritize “shortest” coflow

coflow 1 = (F₁, F₂, F₃, F₄)     coflow 2 = (F₅, F₆, F₇, F₈)
Communication Efficiency

- Minimize the average coflow completion time
- Coflow scheduling: prioritize “shortest” coflow
- Coflow2 is shorter than Coflow1

\[
\text{coflow 1} = (F_1, F_2, F_3, F_4) \quad \text{coflow 2} = (F_5, F_6, F_7, F_8)
\]
MapReduce Runtime

map1,2,3: computation-intensive tasks

reduce1,2,3: network-intensive tasks

Compute
Communicate
Refine
Joint Optimization: Motivation

- **Network-intensive** tasks (reduce 1,2,3) leave the allocated CPU idle.

- **Computation-intensive** tasks (map 4,5,6) need to process raw data 4,5,6, but have to wait for reduce 1,2,3 to finish.

J.Jiang, S.Ma, B. Li, B.Li, *Symbiosis*: Network-Aware Task Scheduling in Data-Parallel Frameworks, INFOCOM 16
Network-aware Scheduling

- **Symbiosis**: co-locate network- and computation-intensive tasks to **balance** the utilization of different resources.

  - e.g., **co-locate** map4, map5, map6 onto the **same** machines before reduce1, reduce2, reduce3 finish

  - Speed up map4, map5, map6

J. Jiang, S. Ma, B. Li, B. Li, *Symbiosis*: Network-Aware Task Scheduling in Data-Parallel Frameworks, INFOCOM 16
Scheduling: Objectives

- Performance
  - Run as many tasks as quickly as possible
- Fairness
  - A datacenter is shared by multiple tenants
A Shared, Multi-tenant Cloud

How should cloud resources be fairly shared?
Dominant Resource Fairness: the *de facto* fairness notion for *multiple resources*
Dominant Resource

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
Dominant Resource

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
Dominant Resource

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
Dominant Resource

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
Dominant Resource

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
Dominant Resource Fairness

Equalize the share of dominant resource

Task of Job 1
<1 core, 3 GB>

Task of Job 2
<2 cores, 1 GB>

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
Dominant Resource Fairness

Equalize the share of dominant resource

<table>
<thead>
<tr>
<th>Memory (GB)</th>
<th>0</th>
<th>3</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td># Cores</td>
<td>0</td>
<td>4</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Task of Job 1: <1 core, 3 GB>
- Task of Job 2: <2 cores, 1 GB>

\[
\frac{8}{12} = \frac{2}{3}
\]

\[
\frac{12}{18} = \frac{2}{3}
\]

Ghodsi et al., Dominant Resource Fairness: Fair Allocation of Multiple Resource Types, NSDI’11
DRF can also be applied to achieve communication fairness.
Communication Fairness

Key intuition: each access link is a type of resource.

Coflow 1 = (F₁, F₂, F₃, F₄)  
Coflow 2 = (F₅, F₆, F₇, F₈)

Coflow1 and coflow2 compete for the bandwidth on four links.

Transform to a multi-resource fairness problem.
Scheduling: Objectives

- Performance
  - Run as many tasks as quickly as possible
- Fairness
  - A datacenter is shared by multiple tenants
However, **performance** and **fairness** are the two **conflicting objectives**
Performance
Performance

“Shortest” task first
or
“Shortest” coflow first
Performance

“Shortest” task first
or
“Shortest” coflow first

favors mice over elephants
Performance

“Shortest” task first
or
“Shortest” coflow first

Fairness

favors mice over elephants
Performance

“Shortest” task first
or
“Shortest” coflow first

Fairness

DRF
or
Max-min fairness

favors mice over elephants
Performance

“Shortest” task first
or
“Shortest” coflow first

favors mice over elephants

Fairness

DRF
or
Max-min fairness

Poor performance
due to low utilization
<table>
<thead>
<tr>
<th>Performance</th>
<th>Fairness</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Shortest” task first or “Shortest” coflow first</td>
<td>DRF or Max-min fairness</td>
</tr>
<tr>
<td>favors mice over elephants</td>
<td>Poor performance due to low utilization</td>
</tr>
</tbody>
</table>
Performance

“Shortest” task first
or
“Shortest” coflow first

favors mice over elephants

Fairness

DRF
or
Max-min fairness

Poor performance due to low utilization

Can we strike a **flexible balance** in-between?
Navigating the Fairness-Efficiency Tradeoff

Efficiency

Fairness

Fairness knob $\alpha$

W. Wang, S. Ma, B. Li, B. Li, Coflex: Navigating the Fairness-Efficiency Tradeoff for Coflow Scheduling, INFOCOM 17
Navigating the Fairness-Efficiency Tradeoff

Efficiency  Fairness

Fairness knob $\alpha$

W. Wang, S. Ma, B. Li, B. Li, Coflex: Navigating the Fairness-Efficiency Tradeoff for Coflow Scheduling, INFOCOM 17
Navigating the Fairness-Efficiency Tradeoff

Efficiency

Fairness

Fairness knob $\alpha$
Navigating the Fairness-Efficiency Tradeoff

Efficiency

Fairness

Fairness knob $\alpha$
Navigating the Fairness-Efficiency Tradeoff

Cloud operator specify a fairness knob $\alpha \in [0, 1]$

\[
\begin{align*}
\text{maximize} & \quad \{x_k\} \quad \sum_k U_k(x_k) \\
\text{subject to:} & \quad \sum_k x_k \leq c \\
& \quad x_k \geq \alpha \cdot \overline{x_k}
\end{align*}
\]

- performance utility
- capacity constraints
- fairness constraints

Each tenant receives at least an $\alpha$-portion of its fair share
Navigating the Fairness-Efficiency Tradeoff
Navigating the Fairness-Efficiency Tradeoff

- Computation
Navigating the Fairness-Efficiency Tradeoff

- Computation
  - CFQ [Chen et al., INFOCOM 17]
Navigating the Fairness-Efficiency Tradeoff

- **Computation**
  - CFQ [Chen et al., INFOCOM 17]
    - Near-optimal performance with isolation guarantee
Navigating the Fairness-Efficiency Tradeoff

- Computation
  - CFQ [Chen et al., INFOCOM 17]
    - Near-optimal performance with isolation guarantee
  - Speculative slot reservation [Chen et al., ICDCS 17]
Navigating the Fairness-Efficiency Tradeoff

- **Computation**
  - CFQ [Chen et al., INFOCOM 17]
    - Near-optimal performance with isolation guarantee
  - Speculative slot reservation [Chen et al., ICDCS 17]
    - Enforcing service isolation for iterative computations
Navigating the Fairness-Efficiency Tradeoff

- **Computation**
  - CFQ [Chen et al., INFOCOM 17]
    - Near-optimal performance with isolation guarantee
  - Speculative slot reservation [Chen et al., ICDCS 17]
    - Enforcing service isolation for iterative computations

- **Communication**: Coflex [Wang et al., INFOCOM 17]
Navigating the Fairness-Efficiency Tradeoff

- **Computation**
  - CFQ [Chen et al., INFOCOM 17]
    - Near-optimal performance with isolation guarantee
  - Speculative slot reservation [Chen et al., ICDCS 17]
    - Enforcing service isolation for iterative computations

- **Communication**: Coflex [Wang et al, INFOCOM 17]

How to navigate the fairness-efficiency tradeoff with the interplay of computation and communication?
Navigating the Fairness-Efficiency Tradeoff

- Computation
  - CFQ [Chen et al., INFOCOM 17]
    - Near-optimal performance with isolation guarantee
  - Speculative slot reservation [Chen et al., ICDCS 17]
    - Enforcing service isolation for iterative computations

- Communication: Coflex [Wang et al., INFOCOM 17]

- How to navigate the fairness-efficiency tradeoff with the interplay of computation and communication?

Open problem
Takeaway

- Cloud is the **foundation** of big data and multimedia applications.

- Various user-related analytics are powered by data-parallel computing with **MapReduce** abstraction.

- Computation and communication scheduling **jointly determine** the performance and the fairness of cloud tenants.