Modeling and Optimization of Big Data Systems

Li Zhang
System Analysis & Optimization Group
IBM T. J. Watson Research Center

Collaborations with many colleagues, students at IBM and many academic collaborators.
Big Data Systems

- Characteristics of Big Data Systems
  - Volume
  - Variety
  - Velocity
  - Variability
  - Veracity
  - Complexity

- Examples
  - Storage: HDFS, GFS, ...
  - Processing: MapReduce, Spark, Hive, ...
  - NoSQL stores:
    - Column: Cassandra, Hbase, ...
    - Document: CouchDB, DocumentDB, MongoDB, ...
    - Key-value: MemcacheDB, Redis, Aerospike, ...
    - Graph: Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, ...
    - Multi-model: Alchemy Database, CortexDB, ...
MapReduce Systems

➤ Commonly used in Big Data analytics
  ▪ By Facebook, Yahoo, Google, TaoBao, ...

➤ Simple workloads
  ▪ Word count, grep, sort, sampling, ...

➤ Complex workloads
  ▪ NutchIndexing, PageRank, Bayesian classification, K-means clustering, log analyzer, simulation, ...
  ▪ SQL like queries (Hive, JaqI, ...) compiled to DAG of MapReduce jobs, ...

➤ Map heavy
  ▪ Word count, grep, sampling, ...

➤ Reduce heavy
  ▪ Sort, queries, ...

➤ Data (I/O) heavy
  ▪ Sort
MapReduce overview

Function View

System View

Process View

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
</tbody>
</table>

Input = set of key/value pairs

Output = set of key/value pairs

Map

Reduce

write

Output File 0

Output File 1

JobTracker

assign map

assign reduce

Input Data

map

reduce

Copy/Shuffle

Time

MASCOTS 2015

<table>
<thead>
<tr>
<th>copy/shuffle</th>
</tr>
</thead>
<tbody>
<tr>
<td>sort/merge</td>
</tr>
</tbody>
</table>

| map |
| reduce |

| time |
Map Reduce Modeling & System Optimization

Goal

- Identify inefficiencies in MapReduce mechanisms & fix them
- Improve the scheduling mechanism
- Performance modeling based approach for capacity planning of MapReduce applications

Our past work

- Worked on both Hadoop & IBM Platform Symphony
- Inefficiencies:
  - Reduce starvation, improve data locality, avoid scheduling delay
- Better scheduling & memory management
  - Pause/resume for reducers, task interleaving
- Performance modeling and capacity planning
  - Benchmarking on Symphony clusters for representative benchmarks
  - Gray box performance models to
    - capture perf metrics (e.g. completion time) as a function of job & system parameters (e.g. data size, cluster size, # of map tasks & reduce tasks, …)
  - Help users determine required capacity setting for a target level SLA
- Deliver capabilities to IBM Cloud products and services

MASCOTS 2015
A scheduler is critical for good performance in presence of multiple jobs
1) >25,000 MapReduce jobs/day (Facebook 2010)
2) Short jobs after large ones (trace study)
Scheduling is Not Easy!

- **Complexities**
  - Multiple phases for each job
  - Multiple resources (cpu, or I/O, or network) may be stressed
  - Fork and join feature for map/copy/shuffle phase
  - Jobs with different characteristics
    - Map heavy, reduce heavy, ...
  - Move computation vs move large amount of data
  - May not be work conserving

- **Many Existing Schedulers (e.g. Fair)**
  - Lack of cooperation between map and reduce tasks
    - Most existing work only optimizes the scheduling of map tasks or reducers independently
  - Dependence between map and reduce causes starvation
  - No consideration of Reduce task locality (which depends on map task allocations)
Existing work

- **Fair (Facebook)**
  - Ensure a minimum number of slots to a job (fair for maps)

- **Quincy and Mantri for Dryad (Microsoft)**
  - Support a graph represented data processing model (DAG – direct acyclic graph)

- **Capacity (Yahoo)**
  - Support for multiple queues each with a fraction of capacity; a job is submitted to a queue

- **Others**
  - LATE – scheduling speculative tasks
  - Delay – improve data locality
  - Deadline oriented schedulers

- **Research at IBM**
  - FLEX – add-on module to Fair to optimize a number of metrics
  - M3R – Main Memory MapReduce engine in X10
  - Platform Computing
Starvation problem

Difference between map and reduce

- **Map**: small and independent, run in parallel
- **Reduce**: long (fetch/shuffle → sort/merge → reduce)
  - Launched in a greedy manner

![Diagram showing map and reduce slots with jobs and time](image)
Real experiment

- If only Job 2’s reduce could start earlier …
Scheduler design

- **Coupling:** launch reduce tasks according to the progress of finished map tasks
  - Reduce starvation

- **Wait scheduling for reduce:** place reducers close to the “centrality” of the intermediate data on the tree topology by skipping some received heartbeats
  - Reduce data movement (improve data locality for reduce)

- **Random peeking for map:** allow launch map task on remote node
  - Avoid scheduling delay (in large clusters)

receive a heartbeat

reduce progress > map progress

yes

schedule reducers
Wait Scheduling

no

schedule mappers
Random Peeking Scheduling

MASCOTS 2015
Coupling to mitigate starvation

**Fair**

Job 2 finishes late

Job 2 starts to wait
For reduce slots

**Coupling**

Job 2 finishes early

Gradually launch reduce tasks

MASCOTS 2015
Starvation time

**Fair**

Long Starvation Time

Starvation Time :=
Average time between completion of last map task and start of each reduce task

**Coupling**

0 Starvation Time

MASCOTS 2015
Reduced starvation

- **Experiment**
  - 22 jobs: map heavy (Grep, QuasiMonteCarlo), reduce heavy (sort), small, large …
  - Repeat 5 times
  - Coupling scheduler reduces starvation time

<table>
<thead>
<tr>
<th>JobID</th>
<th>Job Description</th>
<th>Time</th>
<th>M</th>
<th>R</th>
<th>$S_F$</th>
<th>$S_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Grep [1-5] randomInput</td>
<td>0</td>
<td>148</td>
<td>15</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>02</td>
<td>Grep [5-9] randomInput</td>
<td>30</td>
<td>148</td>
<td>15</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>03</td>
<td>QuasiMonteCarlo</td>
<td>150</td>
<td>5</td>
<td>1</td>
<td>6.1</td>
<td>0.0</td>
</tr>
<tr>
<td>04</td>
<td>WordCount randomInput05</td>
<td>170</td>
<td>8</td>
<td>1</td>
<td>4.0</td>
<td>0.0</td>
</tr>
<tr>
<td>05</td>
<td>Grep [2-6] randomInput05</td>
<td>190</td>
<td>8</td>
<td>2</td>
<td>3.9</td>
<td>0.0</td>
</tr>
<tr>
<td>06</td>
<td>Grep [3-6] randomInput05</td>
<td>210</td>
<td>8</td>
<td>2</td>
<td>2.8</td>
<td>0.0</td>
</tr>
<tr>
<td>07</td>
<td>Grep [4-6] randomInput05</td>
<td>230</td>
<td>8</td>
<td>3</td>
<td>2.8</td>
<td>0.0</td>
</tr>
<tr>
<td>08</td>
<td>Grep [a-h][a-z] wikiInput</td>
<td>470</td>
<td>427</td>
<td>15</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>09</td>
<td>Grep [a-g][a-z] wikiInput</td>
<td>500</td>
<td>427</td>
<td>15</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>Sort randomPair1</td>
<td>800</td>
<td>224</td>
<td>27</td>
<td>4.9</td>
<td>4.5</td>
</tr>
<tr>
<td>11</td>
<td>Grep [1-2] randomInput10</td>
<td>860</td>
<td>15</td>
<td>5</td>
<td>8.3</td>
<td>0.9</td>
</tr>
<tr>
<td>12</td>
<td>Grep [1-5] randomInput05</td>
<td>880</td>
<td>8</td>
<td>3</td>
<td>8.6</td>
<td>0.6</td>
</tr>
<tr>
<td>13</td>
<td>Grep [6-9] randomInput05</td>
<td>900</td>
<td>8</td>
<td>2</td>
<td>8.4</td>
<td>0.2</td>
</tr>
<tr>
<td>14</td>
<td>Sort randomPair3</td>
<td>1020</td>
<td>64</td>
<td>27</td>
<td>5.3</td>
<td>3.5</td>
</tr>
<tr>
<td>15</td>
<td>Grep [3-8] randomInput20</td>
<td>1140</td>
<td>30</td>
<td>2</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>16</td>
<td>WordCount randomInput10</td>
<td>1440</td>
<td>15</td>
<td>1</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>17</td>
<td>Sort randomPair2</td>
<td>1710</td>
<td>352</td>
<td>27</td>
<td>0.3</td>
<td>5.6</td>
</tr>
<tr>
<td>18</td>
<td>QuasiMonteCarlo</td>
<td>2110</td>
<td>15</td>
<td>1</td>
<td>0.3</td>
<td>0.0</td>
</tr>
<tr>
<td>19</td>
<td>Grep [1-5] randomInput05</td>
<td>2125</td>
<td>8</td>
<td>3</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>Sort randomPair3</td>
<td>2245</td>
<td>64</td>
<td>27</td>
<td>0.5</td>
<td>1.1</td>
</tr>
<tr>
<td>21</td>
<td>RandomWriter</td>
<td>2365</td>
<td>150</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>22</td>
<td>QuasiMonteCarlo</td>
<td>2485</td>
<td>10</td>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Processing Time (40% reduction)
Larger, more realistic experiment

Test bed

One master node and 62 slave nodes; each node has 4 cores (2933MHz, 32KB cache size), 6GB of memory and 72GB of disk.
Set 4 map slots and 2 reduce slots for each node.

Compare job processing times distribution (200 jobs)

21.3% improvement in average job processing time
It’s all good and cool and dry so far.

But, what about a MODEL?

One does not get to present at MASCOTS without showing a MODEL!

Let’s get wet!
Queueing model & analysis
Overlapping Tandem Queue

input → map phase → reduce phase → output

- Jobs
  - job2
  - job1
  - job2
  - job3
  - ... 

- Tasks
  - J2task0
  - J1task1
  - J2task1
  - J3task0
  - ...

- Phases
  - Map phase
  - Shuffle phase
  - Reduce phase

- Expressions
  - \( \text{progress}_1(\text{job-i}) \geq \text{progress}_2(\text{job-i}) \)

Different from classical queuing models
Overlapping Tandem Queue

Map work

Reduce work

Map completed work

Reduce completed work
Traditional Tandem Queue

Map work

Reduce work

Map completed work

Reduce completed work

MASCOTS 2015
Which model fits better?

Map work

Reduce work

Measurement from real system

Overlapping Tandem Queue

Traditional Tandem Queue
Difference from classical models

- **Overlapping tandem queue**
  - Fluid model: big data is like fluid!

- **Tandem queueing model**
  - No overlapping
  - Usually Poisson arrivals
  - Usually independent exponential service at each station

- **Flow shop model**
  - No overlapping
  - Batch jobs
  - Focus on non-preemptive scheduling and makespan criterion
Model parameters

Reasonable assumption when each job has more tasks than machines.

\[ \mu_1 = \text{total processing capacity of all map slots} \]
\[ \mu_2 = \text{total network capacity between map slots and reduce slots} \]

Size of \( \text{job-}i: \frac{x_i}{y_i} \) → service time: \( x_i \) for map phase, \( y_i \) for shuffle phase

Can be estimated in practice:

(a) Run a few map tasks of \( \text{job-}i \).
(b) Linear prediction based on task time and intermediate data size.

\[ \text{s.t. } \text{progress}_1(\text{job-}i) \geq \text{progress}_2(\text{job-}i) \]
Online scheduling algorithm: 
MaxSRPT

Focus on “finishing small jobs early”.

Algorithm. Both stations work on the jobs using SRPT based on $\max(x_i(t), y_i(t))$
subject to data availability.

How good is it?

Not more than 2

Theorem. Denote $\alpha = \max_i \max(x_i/y_i, y_i/x_i)$. The algorithm is $2\alpha/(1 + \alpha)$-speed optimal.

Better for “balanced jobs”
Online scheduling algorithm: SplitSRPT

Focus on “keeping the shuffle phase busy”.

Mix map-heavy job and shuffle-heavy job.

**Algorithm.** Denote $\beta = \min_i \max(x_i/y_i, y_i/x_i)$. Split the capacity of the map station so that $\mu_{m1} : \mu_{m2} = 1 : \beta$. Split the capacity of the shuffle station so that $\mu_{s1} : \mu_{s2} = \beta : 1$. For the new arrival $J_k$, update $S_1 = S_1 \cup \{J_k\}$ if $x_k/y_k \geq 1$ and $S_2 = S_2 \cup \{J_k\}$ if $x_k/y_k < 1$. Run jobs in $S_1$ by SRPT (map size) using $\mu_{m2}$ and $\mu_{s2}$. Run jobs in $S_2$ by SRPT (shuffle size) using $\mu_{m1}$ and $\mu_{s1}$.

**Theorem.** Denote $\beta = \min_i \max(x_i/y_i, y_i/x_i)$. The algorithm is $(1 + 1/\beta)$-speed optimal.

Better for “unbalanced jobs” Not more than 2
Queueing model

- $A_i$ – arrival interval
  Poisson process
- $B_i$ – map service of job $i$
  Power law

- $R_i$ – # of reducers of job $i$
  specified by users

**Task progress constraint**

$$\Theta_{C_i^j(t)} \leq \Theta_{B_i(t)}$$

**Diagram:**

- $A_i$: Job submission
- $B_i$: Copy/shuffle
- $C_i^j$: Reduce phase
- $D_i^j$: Available slot

**Equation:**

$$\left(C_i^j, D_i^j\right), 1 \leq j \leq R_i$$

$$\Theta_{C_i^j(t)} \leq \Theta_{B_i(t)}$$
Heavy-tailed workload characteristics

S. Kavulya, et al. *An analysis of traces from a production mapreduce cluster*, CCGRID '10

\[ B_i \] - i.i.d. map service is regularly varying with index \( \alpha \)

\[ P[B_i > x] = \frac{l(x)}{x^\alpha}, \ x \geq 0, \ \alpha > 1 \]

\( l(x) \) is slowly varying

\[ \lim_{x \to \infty} \frac{l(\lambda x)}{l(x)} = 1, \ \lambda > 0 \]

Figure: Power-law distribution of jobs (Facebook) in the number of tasks and input sizes. Power-law exponents are 1.9 and 1.6 when fitted with least squares regression.

G. Ananthanarayanan, et al. *PACMan: Coordinated Memory Caching for Parallel Jobs*, NSDI '12
Criticality phenomenon in heavy tails

If $\lambda E[B] < 1$, $\lambda E[R]E[C] < r$, $\alpha > 1$, then, for Fair Scheduler

1. If $\mathbb{P}[R > r - \lambda E[R]E[C]] > 0$, then

$$\lim_{x \to \infty} \frac{\log \mathbb{P}[T_f > x]}{\log x} = -\alpha + 1.$$ 

A large job temporarily blocks the reduce queue

input rate $> \text{service rate}$

2. If $\mathbb{P}[R < r - \lambda E[R]E[C]] = 1$ and $\alpha > 3$, then

$$\mathbb{P}[T_f > x] \sim \mathbb{P}[B > (1 - \rho)x],$$

implying

$$\lim_{x \to \infty} \frac{\log \mathbb{P}[T_f > x]}{\log x} = -\alpha.$$ 

input rate $< \text{service rate}$

3. If $1 < \alpha \leq k^*/(k^* - 1)$, $k^* \geq 2$ and $\mathbb{P}[r/(k^* - 1) > R \geq r/k^*] = 1$, $C \equiv 0$, then

$$\lim_{x \to \infty} \frac{\log \mathbb{P}[T_f > x]}{\log x} = -k^*(\alpha - 1).$$

$k^*$ large jobs temporarily block the reduce queue
Validation of criticality

- **Test bed**
  - 24 nodes – 4 map + 2 reduce slots
  - Linux 22.6.18-194.17.4.el5 kernel.
  - Four 2.67GHz hex-core per node
  - Intel Xeon X5650 CPUs with Hyper-threading capability
  - 24GB memory + two 500GB Western Digital SATA hard drivers.
  - All nodes on the same Top-of-Rack 1Gigabit Ethernet switch.

- **Results match strikingly well with analysis**
  - Each job under Test 2 runs faster than under Test 1 in a stand-alone environment
  - On contrary, job execution times under Test 2 are much worse than Test 1 in a shared environment

---

**Table: Composition of the job flow that is similar to the Facebook workload**

<table>
<thead>
<tr>
<th>Group</th>
<th>Benchmark</th>
<th>Input Size</th>
<th>Job (#)</th>
<th>ReduceTasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wordcount</td>
<td>64MB</td>
<td>330</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Termvector</td>
<td>128MB</td>
<td>109</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Invertedindex</td>
<td>256MB</td>
<td>36</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Invertedindex</td>
<td>512MB</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Invertedindex</td>
<td>1GB</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>Terasort</td>
<td>2GB</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>Adjacencylist</td>
<td>4GB</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>Sequencecount</td>
<td>8GB</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Sequencecount</td>
<td>16GB</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

- Total Jobs: 506

---

**Graph:**

- $P[T>t] \approx 8.6t^{0.9}$
- $P[T>t] \approx 100.6t^{1.9}$
DynMR for IBM Platform Symphony

Performance issues

Macroscopic
- Difficulty in selecting optimal performance parameters (reduce #, MR ratio, slow-start); Auto-tuning (similar to star-fish) is also difficult
- No flow control for fetching data (cause under/over utilization of the fetch threads)
- Selfish users can monopolize the cluster by running many long reduce tasks; Unfairness to small jobs
- Unfair (YARN emphasizes fairness by assuming a single task type, MapReduce has both map and reduce)

Microscopic
- Long-tailed reduce tasks caused by data skew or heterogeneous computing nodes
- Reducer bundle several functional phases together and can only process the data of one partition; No pipelining between fetchers and mergers
- No pause-resume mechanism for ReduceTasks
How can we do better?

» Wish list:

- Preemption
  ➞ Efficient context switching

- Balance Map & Reduce resources
  ➞ Flow control
DynMR Design

★ Multiple ReduceTasks in a progressive queue share a single JVM

Guiding principle

1. Use fine-grained reduce tasks (smaller partitions)
2. Delicately schedule tasks in refined time scales
3. Efficient task context switching

DynMR adaptively interleaves partially-completed ReduceTasks and backfills MapTasks

When? 1. Detect-and-yield: identify best time points to switch tasks

How many? 2. Flow control: assemble multiple tasks into a progressive queue, to form a “bigger task”

What about data? 3. Segment manager

1. Manage data segments of multiple tasks
2. Extract merge threads as standalone services
DynMR High Level Architecture

Master node

Progressive Queue (C++)
MapTask
ReduceTask

Flow Control
Session Manager (Job Tracker C++)

Slave node

Instance Manager (C++)
Control & Messages

Service Instance

JVM
Heap
Detect & Yield
Segment Manager
Merge
Disk

Service Instance

JVM
Heap
Detect & Yield
Segment Manager
Merge
DynMR Execution Example

Data flow:

- Memory-to-disk merge
- Disk-to-disk merge

JVM 1:
- Map
- Map
- Fetch
- Map
- Map
- Fetch
- Map
- Fetch
- Reduce

JVM 2:
- Map
- Map
- Fetch
- Map
- Map
- Map
- Fetch
- Map
- Fetch
- Reduce

JVM 3:
- Map
- Fetch
- Map
- Map
- Fetch
- Map
- Map
- Fetch
- Reduce

JVM 4:
- Map
- Map
- Map
- Fetch
- Map
- Map
- Map
- Fetch
- Reduce

Available
Segment manager

Fetch threads
Remote map output

Memory

Within a JVM
Standalone services

Time

MASCOTS 2015
Experiments – Single Job

Terasort

Better utilization

Shorter execution time

Table. 5T Terasort configurations

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>io.sort.mb</td>
<td>650</td>
</tr>
<tr>
<td>io.sort.factor</td>
<td>100</td>
</tr>
<tr>
<td>mapred.job.reduce.input.buffer.percent</td>
<td>0.96</td>
</tr>
<tr>
<td>mapred.job.shuffle.merge.percent</td>
<td>0.96</td>
</tr>
<tr>
<td>mapred.job.shuffle.input.buffer.percent</td>
<td>0.8</td>
</tr>
<tr>
<td>mapred.compress.map.output</td>
<td>true</td>
</tr>
<tr>
<td>mapred.map.output.compression.codec</td>
<td>Lz4Codec</td>
</tr>
<tr>
<td>mapred.output.compression.type</td>
<td>BLOCK</td>
</tr>
<tr>
<td>mapreduce.job.intermediated.data.checksum</td>
<td>false</td>
</tr>
<tr>
<td>JVM heap -Xmx -Xms</td>
<td>1000m</td>
</tr>
</tbody>
</table>

Table. Optimal configuration for the original system

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>map/reduce slot ratio</td>
<td>2:1</td>
</tr>
<tr>
<td>mapred.reduce.slowstart.completed.maps</td>
<td>0.25</td>
</tr>
<tr>
<td>mapred.reduce.tasks</td>
<td>256</td>
</tr>
</tbody>
</table>

Default parameter values are: 1:1, 0.05, 384.
Experiments – Single Job

Terasort

Better utilization

Shorter execution time

# service rounds and # tasks in a progressive queue (skewed data)

Small ReduceTasks yield more often

More Small ReduceTasks in 1 progressive queue

5T skewed Terasort with 300 ReduceTasks

5T skewed Terasort with 2000 ReduceTasks

MASCOTS 2015
Experiment – Multiple Jobs

<table>
<thead>
<tr>
<th>Group</th>
<th>Job name</th>
<th>Input data</th>
<th>Task # (m,r)</th>
<th>Job #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Histogram-movies</td>
<td>9.7GB</td>
<td>(75,10)</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>Grep <code>^a[a-z]*</code></td>
<td>31.1GB</td>
<td>(235,15)</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>Histogram-ratings</td>
<td>46.5GB</td>
<td>(359,20)</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>Inverted-index</td>
<td>52.3GB</td>
<td>(390,25)</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Word-count</td>
<td>66.1GB</td>
<td>(498,30)</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Sequence-count</td>
<td>99.3GB</td>
<td>(749,90)</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>Adjancy-list</td>
<td>29.1GB</td>
<td>(313,100)</td>
<td>2</td>
</tr>
</tbody>
</table>
Summary

- MapReduce systems
- Starvation problem
- Solutions:
  - Coupling scheduler
  - Preemption mechanism
  - DynMR
- Performance Models
  - Overlapping Tandem Queue
  - Online scheduling
  - Heavy tail analysis
Research Activities on MapReduce

- **Coupling scheduler for Hadoop**
  - Infocom 2012, Sigmetrics 2012, Infocom 2013, Cloud 2013

- **Pre-emptive scheduler for Hadoop**
  - Efficient pause/resume mechanism for Reduce tasks
  - Improves fairness & reduces execution time
  - ICAC 2013, Sigmetrics 2014

- **DynMR: dynamic task interleave for Symphony**
  - Interleave Map and Reduce tasks for Platform Symphony product
  - Improve Terasort speed to 1.7x
  - EuroSys 2014

- **Online parameter tuning**
  - Adjust parameter settings (cpu, memory) online for better performance
  - HPDC 2014

- **Performance modeling & capacity planning**
  - Overlapping tandem queue model & algorithm analysis
  - IFIP Performance 2013
  - Capacity planning: given a target execution time, determine number & type of VMs, number of reduce tasks per VM
Thank You!