Scheduling Algorithms in MapReduce

Distributed Mind

Karthik Kotian, Jason A Smith, Ye Zhang
Schedule

- Overview of topic (review)
- Hypothesis
- Research paper 1
- Research paper 2
- Research paper 3
- Project software design
Overview of topic

MapReduce

- Process a large set of data in parallel across a distributed cluster of computers
- Maximize the speed at which a task is completed

Scheduling

- Splits a task into smaller jobs and distributes work to the nodes in the cluster
- Utilize the cluster's resources to its maximum capacity
Simulating schedulers

- **FIFO**
  - Oldest job first.

- **Fair-sharing**
  - Every task gets equal share of resources.

- **Capacity**
  - Built over fairness for large cluster size.
    - Queues
    - Priorities
Hypothesis

● FIFO
  ○ Least average execution time of jobs, compared to other two algorithms, when # of tasks < cluster size.

● Fairness
  ○ Least average execution time of jobs, compared to other two algorithms, when cluster size < # of tasks.

● Capacity
  ○ Least average execution time of jobs, compared to other two algorithms, when cluster size > 100.
Approach to test hypothesis

- Simulate MapReduce with each scheduler.
- Change the:
  - Size of the cluster.
  - Number of tasks.
  - Complexity of the tasks.
- Compare schedulers based on average task completion time.
An Adaptive Scheduling Algorithm for Dynamic Heterogeneous Hadoop Systems

**Problem**: Hadoop scheduling algorithm assumes homogeneous clusters and tasks

**Claim**: "To the best of our knowledge, there is no Hadoop scheduling algorithm which simultaneously considers job and resource heterogeneity"

http://www.cas.mcmaster.ca/~downd/cascon11.pdf
The users submit jobs to the system, where each job consists of some tasks. Each task is either a map task or a reduce task.

The Hadoop system has a cluster. The cluster consists of a set of resources, where each resource has a computation unit, and a data storage unit.
Hadoop System Model

Typical scheduler receives two main messages

- New job arrival message from a user
- Heartbeat message from a free resource
Testing Environment

Performance metrics for Hadoop systems
● Locality
● Fairness
● Satisfying the minimum share of the users
● Mean completion time of jobs

Compare against two common schedulers
● FIFO
● Fair-sharing
Novel Contributions

- Adaptive scheduler based on:
  - fairness and dissatisfaction
  - minimum share requirements
  - heterogeneity of jobs and resources
- Reducing communication costs by not necessarily maximizing a task’s distribution
- Reducing search overhead for matching jobs and resources
- Increasing locality
Comparison Results

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Arrival rate Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>$BM_{i,1}$</td>
<td>Smaller jobs have higher arrival rates</td>
</tr>
<tr>
<td>$BM_{i,2}$</td>
<td>Arrival rates are equal for all jobs</td>
</tr>
<tr>
<td>$BM_{i,3}$</td>
<td>Larger jobs have higher arrival rates</td>
</tr>
</tbody>
</table>

Table 5: Experiment benchmarks

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Workload Type</th>
<th>Jobs Included</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_1$</td>
<td>I/O-Intensive$_i$</td>
<td>small, I/O-heavy</td>
</tr>
<tr>
<td>$W_2$</td>
<td>CPU-Intensive$_i$</td>
<td>small, CPU-heavy</td>
</tr>
<tr>
<td>$W_3$</td>
<td>Mixed$_i$</td>
<td>All jobs</td>
</tr>
</tbody>
</table>

Table 4: Experimental workloads
Results - Dissatisfaction

Figure 4: Dissatisfaction performance metric of the algorithms in I/O-Intensive workload
Figure 5: Dissatisfaction performance metric of the algorithms in CPU-Intensive workload
Results - Dissatisfaction

Figure 6: Dissatisfaction performance metric of the algorithms in Mixed workload
Results - Mean Completion Time

Figure 7: Mean Comp. Time performance metric of the algorithms in I/O-Intensive workload
Results - Mean Completion Time

Figure 8: Mean Comp. Time performance metric of the algorithms in CPU-Intensive workload
Results - Mean Completion Time

Figure 9: Mean Comp. Time performance metric of the algorithms in Mixed workload
Usefulness

- Described FIFO and fair-sharing schedulers
- Comparison of a new scheduling algorithm to existing algorithms
- Average completion time as a comparison
- Modeling the MapReduce system
Investigation of Data Locality and Fairness in MapReduce.

- Zhenhua Guo, Geoffrey Fox, Mo Zhou
- Indiana University, Bloomington, IL
- MapReduce 2012 Proceeding of third international workshop on MapReduce and its application date.

Data Locality[2]

- When a node reports that it has idle slots, the master scans the queue to find the task that can achieve the best data locality.
- Master searches for a task whose input data are located on that node.
- If the search fails, it searches for a task whose input data are located on the same rack as the node.
- If the search fails again, it randomly assigns a task.
Problems Addressed[2]

- Fairness and Data Locality conflict with each other when used together.
- Strict Fairness degrades data locality.
- Pure Data Locality results in unfairness of resource usage.
- In a shared cluster, avoid scenario where small number of users overwhelm the resources of whole cluster.
The paper investigates the tradeoff between data locality and fairness cost.

Proposes algorithm *lsap-sched*
- gives optimal data locality by utilizing the well-known problem LSAP.

In addition, it integrates fairness with *lsap-sched* scheduler. This scheduler is called as *lsap-fair-sched*.
Isap-fair-sched Algorithm\cite{2}

- **Input:**
  - $\alpha$ - DLC scaling factor for non data local tasks.
  - $\beta$ - FC scaling factor for tasks that are beyond its group allocation

- **Output:**
  - assignment of tasks to idle map slots

- **Functions:**
  - find the set of nodes with idle slots.
  - find the set of tasks whose input data are stored on nodes with idle slots.
  - calculate $sto$ of all groups.
Isap-fair-sched contd.[2]

- calculate task Fairness Cost matrix (FC).
- calculate Data Locality Cost matrix (DLC).
- \( C = FC + DLC \).
- If \( C \) is not a square matrix then expand it to one.
- \textit{Isap} algorithm is used to find the optimal assignment which is subsequently filtered and returned.

**Fault:**

- Some tasks with bad data locality may be queued indefinitely.
- If the task have been waiting for a long time, reduce FC and DLC so that they could be scheduled at the subsequent scheduling points.
Results\textsuperscript{[2]}

- Reduces Data Locality Cost by 70-90\% compared to Hadoop's *dl-sched*.

Figure: Tradeoff between Fairness and Data Locality

\textsuperscript{#2 - Research Paper #2}
Usefulness\cite{2}

- We chose this paper to understand and investigate data locality scheduler in detail.
- Although we will not be implementing *lsap-fair-sched* for our simulation because of its complexity, its comparison with Fairness and Hadoop's Data Locality Scheduler gave us valuable insights about Fairness Scheduler (which we are using for our simulation) and its limitations.
Improving MapReduce Performance in Heterogeneous Environments

- Hadoop is an open-source implementation of MapReduce.
- Hadoop's performance is closely tied to its task scheduler.
- In practice, the homogeneity assumptions do not always hold. Hadoop's scheduler can cause severe performance degradation in heterogeneous environments.
- Longest Approximate Time to End (LATE) is highly robust to heterogeneity.

http://dl.acm.org/citation.cfm?id=1855744
http://bnrg.eecs.berkeley.edu/~randy/Courses/CS268.F08/papers/42_osdi_08.pdf
Scheduling in Hadoop

Figure 1: A MapReduce computation. Image from [8].
Assumptions in Hadoop's Scheduler

1. Nodes can perform work at roughly the same rate.
2. Tasks progress at a constant rate throughout time.
3. There is no cost to launching a speculative task on a node that would otherwise have an idle slot.
4. A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.
5. Tasks tend to finish in waves, so a task with a low progress score is likely a straggler.
6. Tasks in the same category (map or reduce) require roughly the same amount of work.
How the Assumptions Break Down

1. Nodes can perform work at roughly the same rate.
2. Tasks progress at a constant rate throughout time.

Heterogeneity:
Hadoop assumes that any detectably slow node is faulty. However, nodes can be slow for other reasons.
3. There is no cost to launching a speculative task on a node that would otherwise have an idle slot.

It will be break down when resources are shared

4. A task's progress score is representative of fraction of its total work that it has done. Specifically, in a reduce task, the copy, sort and reduce phases each take about 1/3 of the total time.

This may cause incorrect speculation of reducers.

5. Tasks tend to finish in waves, so a task with a low progress score is likely a straggler.

Even in a homogeneous environment, these waves get more spread out over time due to variance adding up, so in a long enough job, tasks from different generations will be running concurrently.
The LATE Scheduler

(Longest Approximation Time to End)

Strategy:

Always speculatively execute the task that we think will finish farthest into the future.
The LATE Scheduler

1. Previous progress score:
   For a map, the progress score is the fraction of input data read.
   For a reduce task, the execution is divided into three phases, each of which accounts for 1/3 of the score. (copy, sort, reduce)
2. Current progress rate:
   Use progress rate to decide which one is slower.
   It equals to: \( \frac{\text{progress score}}{T} \)
   \( T \) is the amount of time the task has been running for.
   estimation of time to completion: \( \frac{1 - \text{PS}}{\text{PR}} \)
In a typical MapReduce job, it contains three phases: copy, sort, reduce.

The copy phase of reduce tasks is the slowest, because it involves all-pairs communication over the network.

Tasks quickly complete the other two phases once they have all map output.
Previous progress score:
30% reducers finish
the average progress is: $0.3 \times 1 + 0.7 \times (1/3)$
it is around 53%

current estimation = $\left( \frac{1 - \text{progress score}}{\text{progress score}} \right)^T$
When 30% reducers finish their job.
70% reducers may have progress score 1/3.
But, they have different T, which could be used to estimate which ones slower.
Advantages of LATE

1. It is robust to node heterogeneity, because it will relaunch only the slowest tasks, and only a small number of tasks.
2. LATE takes into account node heterogeneity when deciding where to run speculative tasks.
3. By focusing on estimated time left rather than progress rate, LATE speculatively executes only tasks that will improve job response time, rather than any slow tasks.
Figure 3: EC2 Sort running times in heterogeneous cluster: Worst, best and average-case performance of LATE against Hadoop’s scheduler and no speculation.
Results

Figure 5: EC2 Grep running times with stragglers: Worst, best and average-case performance of LATE against Hadoop’s scheduler and no speculation.
Results

Figure 7: Local Sort with heterogeneity: Worst, best and average-case times for LATE against Hadoop’s scheduler and no speculation.
Usefulness

- In reality, most assumptions that we mentioned previously, could not be ensured all the time. This paper help us understand what issue we may face when we implement the scheduler we selected before.

- For the reason of complexity and lack of data, we may not implement it. But, it offer a good reference for us to evaluate the scheduler that we will implement in our project.
Progress since previous report

- Replaced a research paper with a more relevant paper
  - Better aligns with hypothesis
- Heterogeneous vs. homogeneous
- Software structure design for implementation
Software design

- Java SE 6 and Parallel Java Library
- **Application:**
  - MasterNode Class
  - WorkerNode Class
  - Job Class
  - Scheduler Class
- **Input:**
  - Complexity of job (how long it takes to process)
  - Size of the cluster.
  - Size of the queue.
- **Output:** Average time to execute all jobs.
Software design

MasterNode (main method)

initiates

Returns execution time

WorkerNode Object

initiates

Adds task/s

Scheduler

Task
Software design

```
edu.rit.sim.Simulation

+doAfter(dt:double,event:Event): void
+doAt(t:double,event:Event): void
+run(): void
+time(): void

1

0..*

edu.rit.sim.Event

+doAfter(dt:double,event:Event): void
+doAt(t:double,event:Event): void
+perform(): void
+simulation(): Simulation
+time(): void
```
Software design

```
MasterNode
- scheduler: Scheduler
- cluster: Collection<WorkerNode>
+ addJob(j:Job): void

WorkerNode
+ executionRate: int
- numSlots: int

Scheduler

FIFOScheduler
Fairness Scheduler
CapacityScheduler
```
Software design

java MapRedSim <T> <J> <N> <S> <seed>

- **T**: number of trials
- **J**: number of jobs
- **N**: size of cluster
- **S**: scheduler
- **seed**: random seed
Questions?