Scheduling Large Jobs by Abstraction Refinement

The Cloud Scheduling Problem

- Job J is scheduled on a cloud C.
- Every task has to be scheduled on a compute node.
- A good solution should efficiently use the cloud resources and finish the job in reasonable time.

Existing Dynamic Scheduling based Solutions (e.g., Hadoop)

- A compute node is assigned as the JobTracker, and all other compute nodes as TaskTrackers. The client program sends the job to the JobTracker. Several TaskTrackers are registered at the JobTracker. The JobTracker sends the tasks to the TaskTrackers for processing and checks progress.

Our Proposal: Static Scheduling in Clouds

Core idea:
- We assume that we know an estimate of the maximum time required for every task.
- Static scheduling based on task duration estimates.
- Dynamic scheduling techniques to utilize unused intervals.

Challenge:
- Computing optimal schedule is NP hard.
- Most Heuristics: N • T, where N is the number of compute nodes in the cloud and T is the number of tasks in the job.
- N and T are very large for the cloud scheduling problem.

Abstract Refinement Scheduling

- D = Duration
- M = Multiplicity

THE ABS SCHEDULER FISCH

- Create a fixed abstraction of the data center.
- Maintain the free intervals in every abstract node as an inverted index. This allows efficient retrieval of the first n free intervals in an abstract node.
- Choose abstract tasks in topological order.
- Schedule all tasks in one abstract task in the abstract node that allows to finish the tasks at the earliest (greedy schedule).
- If schedule does not meet requirements, refine job abstraction.

THE ABS SCHEDULER BLIND

- Based on the idea of buddy lists.
- Starts with a coarse data center abstraction as a single abstract node.
- Refines the data center abstraction on scheduling every abstract task.
- Avoids too refined data center abstraction using optimizations like view coarsening.

Simulation

Experimental setup:

- Clouds: C1: 2-tier, 1000 nodes; C2: 2-tier, 1600 nodes; C3: 3-tier, 4000 nodes; C4: 3-tier, 8000 nodes.
- Jobs: 1000 tasks with random computation and data transfer requirements.

Comparing AR Schedulers to Hadoop

Experimental setup:

- Cloud: Rent Amazon EC2 m1.xlarge (15 GB RAM, 4 virtual cores, 64-bit).
- Number of compute nodes: 50N (N is the number of instances).

Results:

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Job Latency</th>
<th>Util Latency</th>
<th>Util Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR</td>
<td>0.34</td>
<td>0.32</td>
<td>93%</td>
</tr>
<tr>
<td>MM</td>
<td>1.34</td>
<td>1.95</td>
<td>77%</td>
</tr>
<tr>
<td>FFT</td>
<td>1.89</td>
<td>1.40</td>
<td>78%</td>
</tr>
<tr>
<td>WF</td>
<td>1.57</td>
<td>0.71</td>
<td>62%</td>
</tr>
</tbody>
</table>

Cloud utilization on different clouds after scheduling the sequence of 1000 jobs:

- FISCH: 90% 71% 73% 76%
- BLIND: 85% 80% 77% 79%