Static Scheduling in Clouds

Thomas A. Henzinger    Anmol V. Singh    Vasu Singh
Thomas Wies    Damien Zufferey

IST Austria

ExCape meeting, June 10 2013, Berkley
Cloud computing gives the *illusion* of \( \infty \) (virtual) resources.

Actually there is a finite amount of (physical) resources.

We would like to efficiently share those resources:

1. being able to distinguish high priority (serving customer *now*) from low priority (batch) requests;
2. schedule accordingly.

Therefore, we should be able to *plan ahead* computations.
Dynamic Scheduling: use work queues, priorities, but limited.

Without knowledge of jobs, this is the best you can do.

We need to ask the user for:
- what kind of resources his job require;
- a deadline/priority for his job.

In exchange we can give him an expected completion time.

We can also offer choice. (time is money.)
User Interface

Job Parser Program

User Interface

Job Scheduler

Cloud

Execution Plan

Job Execution Platform

User chosen schedule

Task finish updates

User Interface

Schedules

Job Scheduler

Cloud Representation

Execution Plan

Job Execution Platform

User chosen schedule
Giving incentive to plan in advance

The scheduler returns not one but many possible schedules with different finish times. Use a pricing model to associate a cost to the schedules. Include the “scheduling difficulty” in the cost, give a discount to schedule with later finish time.

Problem: static scheduling is *hard*. Only possible if the scheduler can handle the work load. So we set up to make scheduling cheap(er).
A Job is a directed acyclic task (DAG) of tasks.

- Node are marked with **worst case duration**.
- Edges are marked with **data transfer**.
- Duration and data can be parametric in the input.
Parametric Jobs

User Job

Schema

Connections

Mappers

Reducers

Job Parser

Execution Plan

Task Details

Object Sizes

Database

Input Data Size

Damien Zufferey Static Scheduling in Clouds ExCape meeting 2013
Datacenter as a tree-like graph:

- internal nodes are router;
- leaves are compute nodes (computation speed);
- edges specifies the bandwidth.
Assumption: job and infrastructure regularity
Idea: regularity makes large scale scheduling feasible
How: Using abstraction techniques
Abstraction for jobs:

Group independent tasks as per a topological sort. Merge them into an abstract task.
Abstraction for jobs:

Group independent tasks as per a topological sort. Merge them into an abstract task.
Abstraction for jobs:

Group independent tasks as per a topological sort. Merge them into an abstract task.
Abstraction for jobs:

Group independent tasks as per a topological sort. Merge them into an abstract task.
Abstraction for jobs:

Group independent tasks as per a topological sort. Merge them into an abstract task.
Abstraction for infrastructure:

Merge nodes to according to network topology:

![Diagram of network topology with nodes labeled 1-3 and busy status indicated.]
Abstraction for infrastructure:

Merge nodes to according to network topology:
Abstraction for infrastructure:

Merge nodes to according to network topology:
Experiments, part 1: simulation

Datacenter:

2-tier datacenter
Half of the nodes: speed $x$, Other half: speed $1.5 \times$

On the job side:

Wavefront (WF)  MapReduce (MR)  Matrix Mutil (MM)

FFT Transform (FFT)
Experiments: the cost of abstraction

We then compare Fisch and Blind to a concrete greedy scheduler (baseline) on a sequence of 100 jobs (10-5000 tasks each). Latency is given per tasks.

<table>
<thead>
<tr>
<th>Scheduler</th>
<th>Latency (ms)</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>293</td>
<td>96 %</td>
</tr>
<tr>
<td>Fisch</td>
<td>0.27</td>
<td>92 %</td>
</tr>
<tr>
<td>Blind</td>
<td>0.16</td>
<td>91 %</td>
</tr>
</tbody>
</table>

(a) Baseline

(b) Fisch

(c) Blind
Experiments: scaling

C1: 2000 nodes, 20 per rack
C2: 1600 nodes, 40 per rack
C3: 4000 nodes, 20 per rack
C4: 8000 nodes, 20 per rack
C6: 1000 nodes, 500 per rack
Caution: static scheduling alone will not work.
- Task duration are conservative estimates;
- Variability of the performance of the compute node.

We use static scheduling with backfilling.

Job:
- The jobs are MapReduce jobs doing image transformation.
- Mapper: 8.1 seconds on average, estimate is 40 seconds
- Reducer: Identity operation

Infrastructure:
- Hadoop streaming version 0.19.0
- Amazon EC2 m1.xlarge instances (15GB RAM, 4 cores)
- Number of mappers = 50 * number of instances
Experiments: compared to Hadoop

Observations:

- The Hadoop framework requires large runtime overhead: results in slowdown of the job execution.
- Static scheduling allows to prefetch data, whereas dynamic scheduling does not.
There is an opportunity to apply methods developed to solve computationally hard problem in verification to other area. While preserving a solid theoretical basis.

Questions ?