Firmament – overview and discussion

Łukasz Dudziak
Based on: “Firmament: fast, centralized cluster scheduling at scale” by Ionel Gog et al.
Why scheduling is important?

- Intelligent scheduling may significantly speed up execution time
  - Heterogenous systems
  - Multiple levels of cache (what tasks are better to be run on the same machine?)
- Fair scheduling

Task-by-task placement
- Fast, good for decentralized schedulers
- Does not have “global” knowledge
- Tasks might spend significant time in a queue (unfair?)

Batching placement:
- Always considers entire workload
- New task == rescheduling
- Has ability to pre-empt running tasks
- Slow
Why scheduling is important?

- Intelligent scheduling may significantly speedup execution time
  - Heterogenous systems
  - Multiple levels of cache (what tasks are better to be run on the same machine?)
- Fair scheduling

Task-by-task placement
- Fast, good for decentralized schedulers
- Does not have “global” knowledge
- Tasks might spend significant time in a queue (unfair?)

Batching placement:
- Always considers entire workload
- New task == rescheduling
- Has ability to pre-empt running tasks
- Slow

Let’s optimize it!
Min-cost max-flow

Send certain amount of flow through a network, such as its overall cost is minimized. Formally:

Minimize \( \sum_{(i,j) \in E} \text{cost}_{(i,j)} \text{flow}_{(i,j)} \)

subject to:

\( \forall v \in V \left( \sum_{(v,i) \in E} \text{flow}_{(v,i)} - \sum_{(j,v) \in E} \text{flow}_{(j,v)} = \text{supply}_v \right) \)

\( \forall (i,j) \in E \left( 0 \leq \text{flow}_{(i,j)} \leq \text{capacity}_{(i,j)} \right) \)
- Solve task scheduling with MCMF problem
- Construct a flow network:
  - Each task is a source node with supply 1
  - Each machine is connected to the sink with a zero-cost edge with capacity 1 (so only one task can be assigned to a machine)
  - Create all relevant connections between tasks and machines, in such a way that an edge between a task $T_i$ and a machine $M_i$ has a cost relative to the cost of running the task on the machine.

Source: "Firmament: Fast, Centralized Cluster Scheduling at Scale" Ionel Gog et al.
• Solve task scheduling with MCMF problem
• Construct a flow network:
  • Each task is a source node with supply 1
  • Each machine is connected to the sink with a zero-cost edge with capacity 1 (so only one task can be assigned to a machine)
  • Create all relevant connections between tasks and machines, in such a way that an edge between a task $T_i$ and a machine $M_i$ has a cost relative to the cost of running the task on the machine.

Source: “Firmament: Fast, Centralized Cluster Scheduling at Scale” Ionel Gog et al.
Quincy

- Solution: aggregators

Source: "Quincy: Fair Scheduling for Distributed Computing Clusters" Michael Isard et al.
• Quincy produces good scheduling, but is slow and does not scale...

Source: “Firmament: Fast, Centralized Cluster Scheduling at Scale” Ionel Gog et al.

What can we do about it?
MCMF algorithms overview

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relaxation</td>
<td>$O(M^3CU^2)$</td>
</tr>
<tr>
<td>Cycle canceling</td>
<td>$O(NM^2CU)$</td>
</tr>
<tr>
<td>Cost scaling</td>
<td>$O(N^2M\log(NC))$</td>
</tr>
<tr>
<td>Successive shortest path</td>
<td>$O(N^2U\log(N))$</td>
</tr>
</tbody>
</table>

Source: “Firmament: Fast, Centralized Cluster Scheduling at Scale” Ionel Gog et al.

N – number of nodes
M – number of edges
C – the largest edge cost
U – the largest edge capacity
M > N > C > U

Source: “Firmament: Fast, Centralized Cluster Scheduling at Scale” Ionel Gog et al.
MCMF algorithms overview

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst-case complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relaxation</td>
<td>$O(M^2CU^2)$</td>
</tr>
<tr>
<td>Cycle canceling</td>
<td>$O(NM^2CU)$</td>
</tr>
<tr>
<td>Cost scaling</td>
<td>$O(N^2M\log(NC))$</td>
</tr>
<tr>
<td>Successive shortest path</td>
<td>$O(N^2U\log(N))$</td>
</tr>
</tbody>
</table>

Source: “Firmament: Fast, Centralized Cluster Scheduling at Scale” Ionel Gog et al.

N – number of nodes
M – number of edges
C – the largest edge cost
U – the largest edge capacity
M > N > C > U

But relaxation is better on average!
• Use relaxation as a primary algorithm
• Use incremental version of cost scaling (40% speedup) as fallback
• Return result from whichever algorithm happens to finish first (both are run simultaneously – they are single-threaded)
• Allow for efficient switching between algorithms
• Provide efficient way of updating a flow network and extracting task placement
Results

Source: "Firmament: Fast, Centralized Cluster Scheduling at Scale" Ionel Gog et al.
• 20 times faster placement than Quincy when scheduling tasks over 12,500 machines

• **Very detailed analysis of existing MCMF algorithms**

• Generic system which allows user to specify custom policies
  • e.g. network-aware policy

• But... it was shown that a policy can have direct impact on MCMF solver’s performance (e.g. load-spreading policy is bad for *relaxation*)
  • The question if the optimizations presented by the authors work well with other policies is left unanswered.
The end

Thank you for attention.