Resource Co-Allocation for Large-Scale Distributed Environments

Claris Castillo (IBM T.J. Watson Research Center)
George Rouskas (North Carolina State University)
Khaled Harfoush (North Carolina State University)
Motivation (1)

- Co-allocation of resources: allocation of multiple resources within the same time window
- Emergence of new paradigms
  - On Demand computing (Amazon EC2, SalesForce, IBMCloud)
- Requirements
  - QoS/SLA support
  - Efficiency
  - Scalability
- Emergence of new applications that capitalized on the availability of distributed computing to perform tasks with spatial and temporal dependencies (MapReduce/financial apps.)
Motivation (2)

- **Applications**
  - Virtual Computing Lab (VCL)
  - MapReduce framework (Hadoop)
  - Grid lambda scheduling
  - Workflow scheduling
Background

- **Naïve Approach**: A co-allocation request can be treated as a group of sequential scheduling requests
  - Inappropriate for time sensitive applications

- **Batch scheduling**
  - Resource driven: optimizing for system performance
    - Limited support for QoS by means of backfilling and priorities
  - Job driven: optimizing for application performance

- **Advance reservations**
  - QoS provisioning
  - Workflow support
  - Multiple drawbacks
Goals

- Providing users with time guarantees by scheduling jobs as they arrive without promoting resource fragmentation
- Allowing better scheduling decisions by keeping look ahead until the horizon of the schedule in a way that is efficient

Contributions

- A co-allocation scheduling algorithm
  - Effective in co-allocating resources and provides support for advance reservations and range search
- Range search
  - Ability of the system to find a set of resources available within a given time window
  - Enable selection and scheduling algorithms that are application specific
- Efficient data structure to organize resource availability
  - Leading to the design of an algorithm that allows a single search operation to identify all required resources efficiently
Problem Description

- $R_2 = (17,17,29,2)$
- $R_1 = (17,17,29,2)$

Diagram:

- $S_1$: A
- $S_2$: C
- $S_3$: E
- $S_4$: J

Events:
- $t_0$, $t_1$, $t_4$, $t_7$, $t_{16}$, $t_{17}$, $t_{18}$, $t_{25}$, $t_{29}$, $t_{30}$, $t_{37}$, $t_{39}$, $t_{42}$
System Model

- We consider the following settings:
  - Scheduler $S$
  - $N$ servers
  - Reservation request $r$ requires service
  - Request $(q_r, s_r, l_r, n_r)$
    - $q_r$: request time
    - $s_r$: earliest time the reservation is needed
    - $l_r$: temporal size of the request (duration)
    - $n_r$: spatial size of the request (no. or servers)
  - Idle period $(st_i, et_i, id_i)$
    - $st_i$: starting time
    - $et_i$: ending time
    - $id_i$: server offering the idle period
Time space is partitioned into time slots of equal length

Idle periods are stored in each time slot they span over

Algorithms searches only into the time slot containing $s_r$

Upon failure to schedule: $s_r = s_r + \Delta t$

Honor atomicity of the request by means of temporal counters

Number of idle periods per time slot can be bounded to $N$ if time slot size is set to the minimum temporal size
Two feasibility criterion: $s_{t_i} < s_{t_x}$ and $e_{t_i} > e_{t_x}$
Performance Evaluation

- **Real workloads drive simulations** [ParArch]

<table>
<thead>
<tr>
<th>Workload</th>
<th>No. of processors</th>
<th>No. of jobs</th>
<th>Avge. length (hrs)</th>
<th>Avge. spatial size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC</td>
<td>512</td>
<td>39,734</td>
<td>5.82</td>
<td>9.48</td>
</tr>
<tr>
<td>KTH</td>
<td>128</td>
<td>28,481</td>
<td>2.46</td>
<td>7.67</td>
</tr>
<tr>
<td>HPC2N</td>
<td>240</td>
<td>202,825</td>
<td>4.72</td>
<td>6.56</td>
</tr>
</tbody>
</table>

- **Two experiments**
  - Comparison to batch scheduling
  - Impact on performance of advance advance-reservations

Temporal-size Penalty

Penalty ($P_r$) vs. Temporal-size ($l_r$) (hours)

Graph showing the penalty vs. temporal-size for two scenarios: KTH-online and KTH-batch. The penalty decreases as the temporal-size increases, with fluctuations in the KTH-online scenario more pronounced than in the KTH-batch scenario.
Waiting Time Distribution

Frequency

Waiting Time $W_r$ (Hours)

CTC-online
CTC-batch
KTH-online
KTH-batch
Waiting time distribution as a function of spatial size
Avg Waiting Time vs. fraction of advance reservations ($\rho$)
Number of operations vs. fraction of advance reservations ($\rho$)
Number of retrials vs. spatial size

<table>
<thead>
<tr>
<th>Workload/n</th>
<th>(0:50]</th>
<th>(50:100]</th>
<th>(100:150]</th>
<th>(150:200]</th>
<th>(350:400]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTC</td>
<td>2.96</td>
<td>5.34</td>
<td>7.22</td>
<td>13.25</td>
<td>127.44</td>
</tr>
<tr>
<td>(No. of retrials)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KTH</td>
<td>10.27</td>
<td>60</td>
<td>120</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>(No. of retrials)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Larger spatial size distribution results in larger number of attempts.
- Temporal size distribution of KTH shows large proportion of small jobs.
Discussion of Results

- Our algorithm can efficiently co-allocate resources while supporting advance reservations.
- Online advance reservations mechanisms might offer a better solution to the problem of co-allocating resources as compared to conventional batch scheduling.
- Our work can be easily extended to support deadlines.
Future Work

- Implement the co-allocation algorithm proposed in the context of
  - Hadoop
  - End-to-end path problem in Grid lambda scheduling
- Impact of workload characteristics on system/user performance
- Uncertainty of completion times
Thank you!

I can be reached by email at:

claris@us.ibm.com