Scheduling Parallel DAG Jobs Online

Ben Moseley (CMU)

Joint work with:
Kunal Agrawal (WahsU)
Jing Li (NJIT)
Kefu Lu (WashU/CMU)
Client-Server Scheduling

- Clients send *parallel* jobs to the server
- Jobs schedule on *identical* processors/machines
- Server processes jobs and provides service guarantees
- Jobs arrive over time – *online*
- Jobs can be *preempted*
- Worst case setting
Service Guarantees

- Flow time – difference between arrival and completion of a job
- Common objectives in online scheduling:
  - Average/Sum Flow Time
  - Maximum Flow Time
  - Throughput
Parallelism Models

• Speed-up Curves
  • Jobs associated with speed-up functions

• Directed-Acyclic Graph (DAG) Model
  • Jobs have work which correspond to a DAG
    • Each job is modeled as a DAG
    • Job completed when last node of its DAG is completed
  • Processing rate depends on the number of nodes being worked on
Parallelism Models

• **Speed-Up Curves**
  - Jobs have total work $W$ divided into phases
    - Each phase has work
    - Phases are processed sequentially
  - Processing rate function $\Gamma(m)$
    - Function of number of processors given
    - Function is usually positive sub-linear
    - Function can be different depending on the phase the job is currently in.

A Job's Phases
Directed Acyclic Graph Model of Parallelism

- Nodes represent computation
- Arrows represent dependencies
Online Study of Models

- DAG model
  - Well-studied *offline*
  - Only studied recently online

- Naturally captures programs generated by languages and libraries such as Cilk, Cilk Plus, Intel TBB, OpenMP.

- Used by applied communities: Cyber-Physical-Systems (Real-Time) community excited (Outstanding paper award ECTRS 2013, Best-Student-Paper Award RTSS 2011)
Results

First results for average flow in DAG model

**Average Flow Time [SODA 2016]**
- LAPS is \((1+\varepsilon)\) speed \(O(1)\) competitive, for fixed \(\varepsilon>0\)
- Best theoretically possible

**Throughput [LATIN 2018]**
- A \((1+\varepsilon)\) speed \(O(1)\) competitive algorithm, for fixed \(\varepsilon>0\)
- Best theoretically possible

**Maximum Flow time [SPAA 2016]**
- A \((1+\varepsilon)\) speed \(O(1)\) competitive algorithm, for fixed \(\varepsilon>0\)
- Open if speed is needed
- Algorithm is **practical**
Algorithm Development

- DAG model has been popular because of its connection to practice
  - Well studied for scheduling a single DAG job to minimize makespan

- Work stealing algorithm: good practical and theoretical performance
  - Used in numerous systems for scheduling a parallel job
  - Non-clairvoyant
  - Distributed protocol
  - No preemption

- Want to emulate this success and use theory for FIFO to guide a modification of Work-Stealing
Work-Stealing

double ended queues

push
Core 1

pop
Core 2

Core 3

Stolen work

Steal
Example: FIFO

FIFO: Execute available nodes of job(s) with earliest arrival

Could be more than one job depending on ready nodes
FIFO: Implementation Challenges

Job 1 arrives at time 0

Job 2 arrives at time 1
FIFO: Implementation Challenges

A global queue $Q$

- storing all available nodes

Core 1

Core 2

Core 3

Job 1 arrives at time 0

Job 2 arrives at time 1
FIFO: Implementation Challenges

A global queue $Q$

- storing all available nodes

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Job 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Job 1 arrives at time 0
Job 2 arrives at time 1
FIFO: Implementation Challenges

A global queue $Q$
- storing all available nodes

Each core at each time step
- executes one node in $Q$
  - from the job with the earliest arrival time

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Core 1" /></td>
<td><img src="image2" alt="Core 2" /></td>
<td><img src="image3" alt="Core 3" /></td>
</tr>
</tbody>
</table>

Job's arrival time
- $0$  $0$  $0$  $1$

$Q$
- ![Queue](image4)
FIFO: Implementation Challenges

A global queue $Q$
- storing all available nodes

Each core at each time step
- executes one node in $Q$ from the job with the earliest arrival time

Core 1

Core 2

Core 3

Job 1 arrives at time 0

Job 2 arrives at time 1

Job’s arrival time

1 0

$Q$
Work Stealing for Multiple jobs

(1) Each core has a queue and executes work from it.

(2) Only when the local queue runs out of work, a core will admit a job from global queue.

(3) Algorithm can steal for other queues or from the global queue.

*Has the same theoretical guarantees as FIFO and gave good practical performance*
Conclusion

• New results for scheduling DAG jobs online
• Results have lead to practically usable algorithms for minimizing maximum flow time
• Recent results submitted for average flow time
  • Much harder due to the need for preemptions

• Open Questions:
  • Is resource augmentation needed for maximum flow time in the DAG and speed up curve model (knowing parallelism)?
  • Practical algorithm for throughput maximization?
**Thank You!**

Questions?

<table>
<thead>
<tr>
<th>Workload</th>
<th>Max Flow Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bing workload</strong></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>0.01</td>
</tr>
<tr>
<td>steal-k-first</td>
<td>0.02</td>
</tr>
<tr>
<td>admit-first</td>
<td>0.03</td>
</tr>
<tr>
<td><strong>Finance workload</strong></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>0.02</td>
</tr>
<tr>
<td>steal-k-first</td>
<td>0.03</td>
</tr>
<tr>
<td>admit-first</td>
<td>0.04</td>
</tr>
<tr>
<td><strong>Log-normal workload</strong></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>0.03</td>
</tr>
<tr>
<td>steal-k-first</td>
<td>0.04</td>
</tr>
<tr>
<td>admit-first</td>
<td>0.05</td>
</tr>
</tbody>
</table>

(a) Bing workload  (b) Finance workload  (c) Log-normal workload