Work Stealing for Interactive Services to Meet Target Latency

Jing Li*, Kunal Agrawal*, Sameh Elnikety†, Yuxiong He†, I-Ting Angelina Lee*, Chenyang Lu*, Kathryn S. McKinley†
*Washington University in St. Louis
†Microsoft Research

* This work and was initiated and partly done during Jing Li’s internship at Microsoft Research in summer 2014.
Interactive services must meet a target latency

**Interactive services**

- Search, ads, games, finance
- Users demand responsiveness
Interactive services must meet a target latency

Interactive services
Search, ads, games, finance
Users demand responsiveness

Problem setting
Multiple requests arrive over time
Each request: parallelizable

Latency = completion time – arrival time
Its latency should be less than a target latency $T$

Goal: maximize the number of requests that meet a target latency $T$
Latency in Internet search

- In industrial interactive services, thousands of servers together serve a single user query.
- End-to-end latency $\geq$ latency of the slowest server

end-to-end response time ($\sim 100$ms for user to find responsive)

**Diagram:**
- Parsing a search query
- Doc lookup & ranking
- Doc lookup & ranking
- Doc lookup & ranking
- Result aggregation & snippet generation
- Target latency
Goal — Meet Target Latency in Single Server

Goal – design a scheduler to maximize the number of requests that can be completed within the target latency (in a single server)
Sequential execution is insufficient

Large request must execute in parallel to meet target latency constraint

Target latency

Request Sequential Execution Time (ms)
(work)
Full parallelism does not always work well

Target latency: 90ms
Full parallelism does not always work well

Target latency: 90ms

**Case 1**: 1 large request + 3 small requests
Full parallelism does not always work well

Target latency: 90ms

**Case 1**: 1 large request + 3 small requests

- Small requests are waiting
- Miss 2 requests

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>130</td>
<td>150</td>
</tr>
</tbody>
</table>

Finish by time 90

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

Finish by time 110
Full parallelism does not always work well

Target latency: 90ms

**Case 1**: 1 large request + 3 small requests

- Core 1: 90ms, 110ms, 150ms
- Core 2: 60ms, 60ms, 60ms
- Core 3: 50ms, 110ms

- Finish by time 270
- Finish by time 110
- Miss 2 requests
- ✔ Miss 1 request
Some large requests require parallelism

Target latency: 90ms

**Case 2**: 1 large request + 1 small request

Finish by time 90

Finish by time 110
Some large requests require parallelism

Target latency: 90ms

**Case 2**: 1 large request + 1 small request

- **Miss 0 request**
  - Core 1: 90
  - Core 2: 90
  - Core 3: 110
  - Finish by time 110

- **Miss 1 request**
  - Core 1: 80
  - Core 2: 80
  - Core 3: 270
  - Finish by time 270
Strategy: adapt scheduling to load

Case 1

Cannot afford to run all large requests in parallel

Case 2

Do need to run some large requests in parallel
Strategy: adapt scheduling to load

**High load** run large requests sequentially

Cannot afford to run all large requests in parallel

**Low load** run all requests in parallel

Do need to run some large requests in parallel
Why does the adaptive strategy work?

Latency = Processing Time + Waiting time

**At low load, processing time dominates latency**
- Parallel execution reduces request processing time
- All requests run in parallel

**At high load, waiting time dominates latency**
- Executing a large request in parallel increases waiting time of many more later arriving requests
- Each large request that is sacrificed helps to reduce waiting time of many more later arriving requests
Challenge: which request to sacrifice?

**Strategy:** when load is low, run all requests in parallel; when load is high, run large requests sequentially
Strategy: when load is low, run all requests in parallel; when load is high, run large requests sequentially.

Challenge 1: non-clairvoyant
   - We do not know the work of a request when it arrives.

Challenge 2: no accurate definition of large requests
   - Large is relative to instantaneous load.
**Strategy:** when load is low, run all requests in parallel; when load is high, run large requests sequentially

**Challenge 1** non-clairvoyant
- We do not know the work of a request when it arrives

**Challenge 2** no accurate definition of large requests
- Large is relative to instantaneous load
- load = 10, large request > 180ms
- load = 20, large request > 80ms
- load = 30, large request > 20ms
Contributions

Tail-control scheduler

Tail-control offline threshold calculation

Tail-control online runtime
Contributions

Tail-control scheduler

Input

Tail-control offline threshold calculation

Target latency $T$
Request work distribution
Available in highly engineered interactive services
Request per second (RPS)

Tail-control online runtime
Contributions

Tail-control scheduler

Input

Tail-control offline threshold calculation

Compute a large request threshold for each load value

Large request threshold table

Tail-control online runtime
Contributions

Tail-control scheduler

Input

Tail-control offline threshold calculation

Large request threshold table

Tail-control online runtime

Use threshold table to decide which request to serialize
Contributions

We modify work stealing to implement tail-control scheduling using Intel Thread Building Block.

![Graph showing target latency miss ratio across different target latencies with baseline 1, baseline 2, and tail-control categories.]

Better performance
Contributions

Tail-control scheduler

Tail-control offline threshold calculation

Large request threshold table

Tail-control online runtime

Implementation details in the paper
Tail-control scheduler

Runtime functionalities:

- Execute all requests in parallel to begin with
- Record total amount of computation time spent on each request thus far
- Detect large requests based on the current threshold and current processing time
- Serializes large requests to limit their impact on other waiting requests
Work Stealing for Single Request

- Workers’ local queues
  - Execute work, if there is any in local queue
  - Steal

![Diagram showing workers and tasks]

- Workers
- Task A
- Execution connections
- Parallelization

1. Execute
2. Steal
3. Parallelize
Generalize Work Stealing to Multiple Req.

- Workers’ local queues + a global queue
  - Execute work, if there is any in local queue
  - Steal – further parallelize a request
  - Admit – start executing a new request

![Diagram showing parallelizable requests arriving at the global queue, workers executing, admitting, and parallelizing requests.](image-url)
Implement Tail-Control in TBB

- Workers’ local queues + a global queue
  - Execute work, if there is any in local queue
  - Steal – further parallelize a request
  - Admit – start executing a new request

- Steal-first (try to reduce processing time)
- Admit-first (try to reduce waiting time)
- Tail-control
  - Steal-first + long request detection & serialization
Evaluation

- Various request work distributions
  - Bing search
  - Finance server
  - Log-normal
- Different request arrival
  - Poisson
  - Log-normal
- Each setting: 100,000 requests, plot target latency miss ratio
- Two baselines (generalized from work stealing for single job)
  - Steal-first: tries to parallelize requests and reduce proc time
  - Admit-first: tries to admit requests and reduce waiting time
Improvement in target latency miss ratio

Better performance

Target latency miss ratio

- 2.5%
- 2.0%
- 1.5%
- 1.0%
- 0.5%
- 0.0%

Target Latency (ms)

- 31.3
- 36.6
- 40.8
- 43.6
- 46.8

Bing workload
Heavy load
(RPS 1200, Util 68.8%)

Hard ➔ Easy to meet the target latency
Improvement in target latency miss ratio

Better performance

Bing workload
Heavy load
(RPS 1200, Util 68.8%)

Target latency miss ratio

steal-first
admit-first
tail-control

Admit-first wins
Steal-first wins

Hard Easy to meet the target latency

Relative load: high low
Improvement in target latency miss ratio

Better performance

Bing workload
Heavy load
(RPS 1200, Util 68.8%)

<table>
<thead>
<tr>
<th>Target Latency (ms)</th>
<th>target latency miss ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.3</td>
<td>2.5%</td>
</tr>
<tr>
<td>36.6</td>
<td>2.0%</td>
</tr>
<tr>
<td>40.8</td>
<td>1.5%</td>
</tr>
<tr>
<td>43.6</td>
<td>1.0%</td>
</tr>
<tr>
<td>46.8</td>
<td>0.5%</td>
</tr>
</tbody>
</table>

- Steal-first
- Admit-first
- Tail-control

<table>
<thead>
<tr>
<th>Imp. over SF</th>
<th>52%</th>
<th>49%</th>
<th>52%</th>
<th>56%</th>
<th>58%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imp. over AF</td>
<td>31%</td>
<td>34%</td>
<td>45%</td>
<td>54%</td>
<td>65%</td>
</tr>
</tbody>
</table>
The inner workings of tail-control

![Graph showing request latency over time with target latency line]

- ○ steal-first
- ♦ tail-control

Target Latency
The inner workings of tail-control

Tail-control sacrifices few large requests and reduces latency of many more small requests to meet target latency.
The inner workings of tail-control

Tail-control sacrifices few large requests and reduces latency of many more small requests to meet target latency.
The inner workings of tail-control

Tail-control sacrifices few large requests and reduces latency of many more small requests to meet target latency.
Tail-control performs well with inaccurate input

Log-normal workload
Heavy load
(RPS 1200, Util 66%)

Target latency (ms) 28.3

Target latency miss ratio

• steal-first
• admit-first
• TC (accurate)
Tail-control performs well with inaccurate input

Slightly inaccurate input work distribution is still useful

- Log-normal workload
  Heavy load
  (RPS 1200, Util 66%)

Target latency miss ratio

- less → more inaccurate input work distribution

- steal-first
- admit-first
- TC (accurate)
- TC low inaccuracy
- TC med inaccuracy
- TC high inaccuracy

Target latency (ms) 28.3
Related work

Parallelizing single job to reduce latency

- [Blumofe et al. 1995], [Arora et al. 2001], [Jung et al. 2005], [Ko et al. 2002], [Wang and O’Boyle 2009], …

Interactive server parallelism optimizing for mean response time and tail latency

- [Raman et al. 2011], [Jeon et al. 2013], [Kim et al. 2015], [Haque et al. 2015]

Theoretical results of server scheduling for sequential and parallel jobs optimizing for mean and maximum response time

- [Chekuri et al. 2004], [Torng and McCullough 2008], [Fox and Moseley 2011], [Becchetti et al. 2006], [Kalyanasundaram and Pruhs 1995], [Edmonds and Pruhs 2012], [Agrawal et al. 2016]
Take home

- For non-clairvoyant interactive services, work distribution is helpful for designing schedulers.
- Given the work distribution, we can devise an offline threshold calculation algorithm to compute large request threshold for every value of instantaneous load.
- We have developed an adaptive scheduler that serializes requests according to the threshold table and demonstrated that it works well in practice.