Topic preview: Scheduling

Siddhartha Sen
Microsoft Research NYC
Scheduling

• Tells you how and when to assign tasks to processors
  • Task = anything you can execute
  • Processor = anything you can execute it on
Scheduling

• Tells you how and when to assign tasks to processors
  • Task = runnable thread, web request, MapReduce job
  • Processors = CPU core, service backend, cluster machine

web request → service backend → worker thread → kernel thread → CPU core
Scheduling

• Tells you how and when to assign **tasks** to **processors**
  • Task = runnable thread, web request, MapReduce job
  • Processors = CPU core, service backend, cluster machine

• Many different settings
  • Within a machine, across machines
  • Single-tiered, multi-tiered
  • Online, offline

• Several different objectives
  • Latency, throughput
  • Priority, fairness
Theoretical view

... task queue processor
Theoretical view
Theoretical view

arrival process
(e.g., Poisson)
Theoretical view

...  

<table>
<thead>
<tr>
<th>task size</th>
<th>service time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(e.g., exponential dist.)</td>
</tr>
</tbody>
</table>
Theoretical view

task priority
Theoretical view

scheduling policy
- first-come-first-served
- shortest-task-first
- priority
Theoretical view

...  

context switch
(by dispatcher)

fair sharing

preemptive?
Theoretical view

scheduling policy
- round-robin (time quantum)
- shortest-remaining-time-first*
- priority

* related to work on progress indicators
Theoretical view
Theoretical view

- fixed-sized pool/
dynamic allocation
- blocking/
polling
- synchronous/
asynchronous
- processor affinity
  (cache locality, NUMA)
Theoretical view

queue network
Takeaways?

• Lots of terms, organized by queueing theory

• Theory makes simplifying assumptions
  • Example: Shortest-task-first is “optimal” because it yields lowest average latency

• In practice no single scheduling policy is best
  • Example: Shortest-task-first needs accurate a priori estimate of task size
    Also, what about tail latency?

• There are many practical issues:
  • How is work dispatched to processors?
  • How do processors wait for work?
  • What is a “processor”? How is it allocated?
    • E.g.: kernel thread managed by Linux scheduler
These practical issues matter

From TAM paper
Session focuses on practical issues

- **Arachne**: Scheduling short tasks on a virtual resource (threads) leads to poor utilization/performance

- **TAM**: Several common problems (e.g., hidden contention, ordering constraints) prevent systems from being schedulable

- **uTune**: Right threading model for multi-tiered microservices depends critically on load

- **RobinHood**: Tail latency depends on request structure and latency of backend services out of our control
Running example: request in multi-tier system

From RobinHood paper

From uTune paper
Running example: request in multi-tier system
Problem observed by Arachne

Busy-waiting for c,b,d,e wastes CPU
(e.g., write replication in RAMCloud)
Problem observed by uTune

Polling at high load wastes CPU; blocking at low load causes expensive wakeups
Problem observed by TAM

Hidden contention causes unfairness

Ordering constraint $b \rightarrow d$ reduces scheduling options
Problem observed by RobinHood

Latency depends on request structure – i.e., $\max(b+d, c, e)$ – which is typically ignored.

Backend causing tail latency is unpredictable, time-varying.
Problem observed by all papers

Cache misses/locality affect tail latencies!
Solution: Arachne

- Adaptively and exclusively assign CPU cores to applications; let app schedule user threads
- Designed to minimize cache misses (e.g., no ready queues)
- Improved Memcached, RAMCloud
Solution: TAM

• Automatically generate and visualize Thread Architecture Model (TAM) of system

• Use TAM to identify (and address) common problems preventing schedulability

• Improved HBase
Solution: uTune

- Taxonomize threading models to understand effect on tail latency
- Dynamically switch threading model based on load
- Improved services from uSuite
Solution: RobinHood

• Explicitly identify backends contributing to tail latency
• Dynamically reallocate cache resources from cache-rich to cache-poor (those causing tail)
• Improved OneRF production system at Microsoft
Thanks!
(And please attend the Scheduling session)