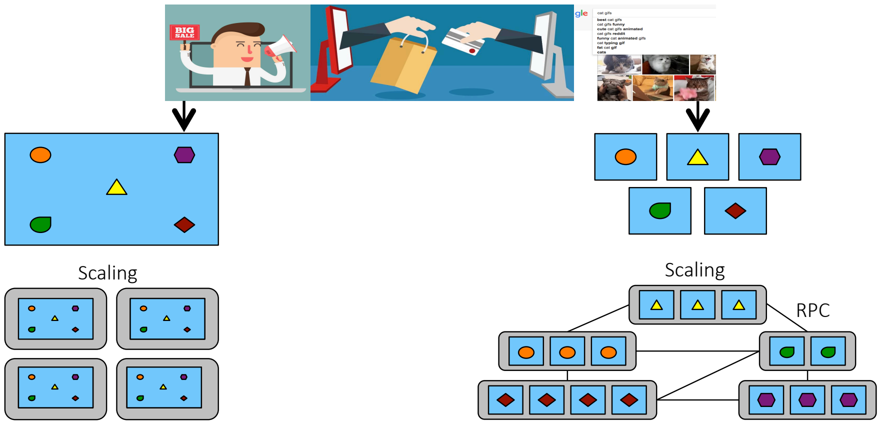
µTune: Auto-Tuned Threading for OLDI Microservices

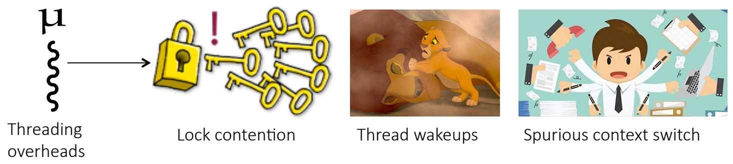
Akshitha Sriraman, Thomas F. Wenisch
University of Michigan

On-Line Data Intensive Applications



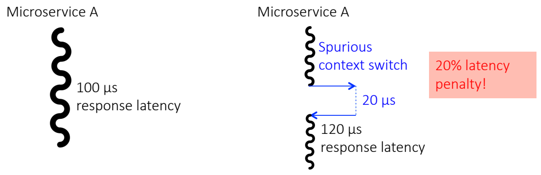
Monoliths (>100 ms SLO) ⟶ Microservices (sub-ms SLO)

Impact of Threading-Induced Overhead

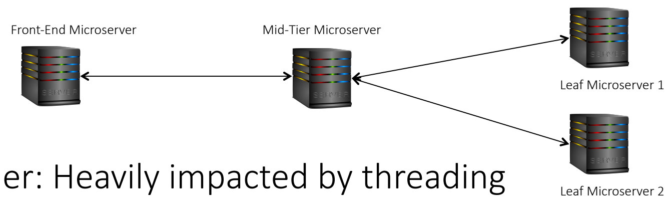


Impact: Minor for monoliths & major for microservices

Example:



Threading Impact on Mid-Tier



Mid-Tier: Heavily impacted by threading

- Server & client
- Fans queries to many leaves
- RPC layer interactions dominate compute

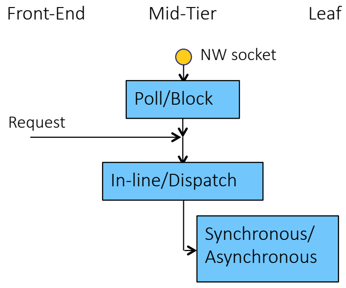
Contributions

- A taxonomy of threading models
  - Structured understanding of threading implications
    - Reveals tail inflection points across load
    - Peak load-sustaining model is subpar at low load
- µTune:
  - Uses tail inflection insights to optimize tail latency
  - Tunes model & thread pool size across load
  - Simple interface: Abstracts threading from RPC code

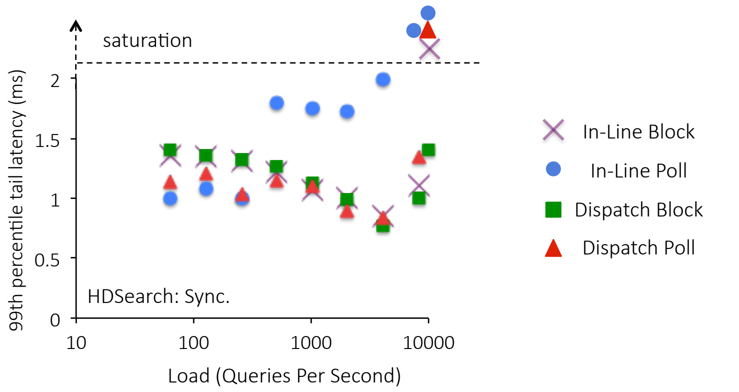
Taxonomy of Threading Models

Threading dimensions:

- Block vs. Poll
- In-line vs. Dispatch
- Sync. vs. Async.



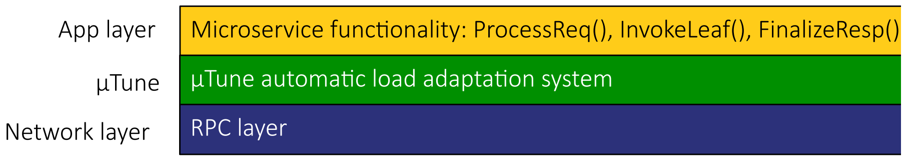
Taxonomy Characterization



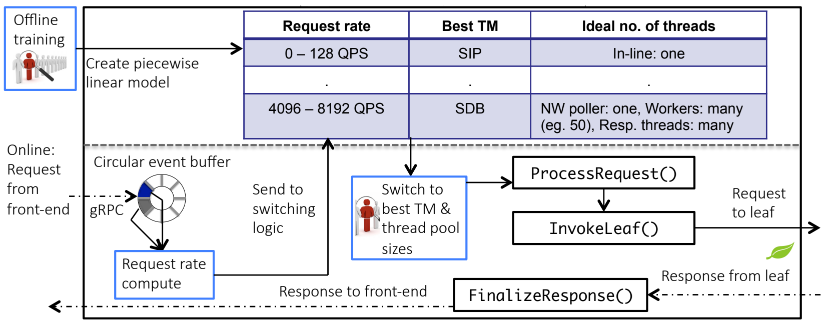
No single threading model works best at all loads

µTune: Automatic Load Adaptation

Abstracts threading boiler-plate code from RPC code

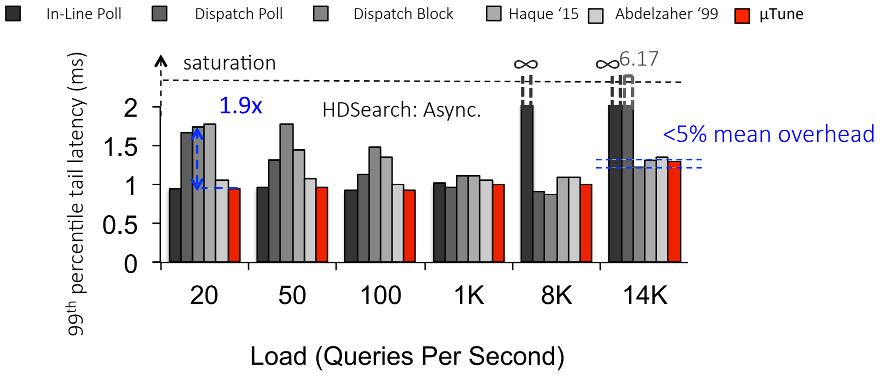

App layer
Microservice functionality: ProcessReq(), InvokeLeaf(), FinalizeResp()
µTune
µTune automatic load adaptation system
Network layer
RPC layer

System design: offline training + run-time adaptation



Request rate	Best TM	Ideal no. of threads
0 – 128 QPS	SIP	In-line: one
.	.	.
4096 – 8192 QPS	SDB	NW poller: one, Workers: many (eg. 50), Resp. threads: many

Result: µTune Under Steady-State Load



- µTune converges to best threading model and thread pool size to improve tail latency by up to **1.9x** over static peak load-sustaining threading model with < 5% mean overhead

Acknowledgement

NSF intel SRC ada Applications Driving Architectures