Performance Engineering for Cloud Microservices

Haoran Qiu
UIUC System Reading Group
Feb 12, 2020

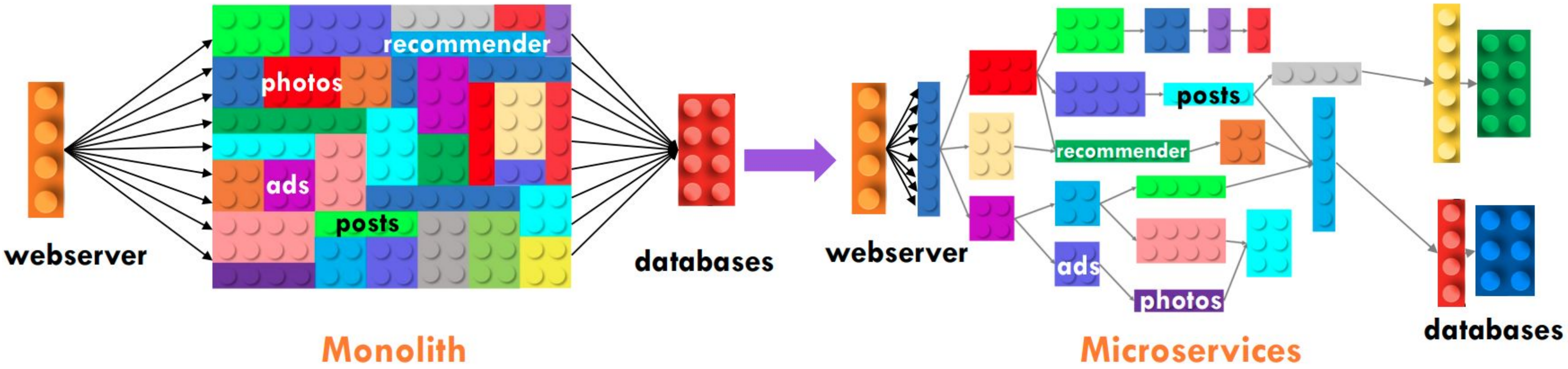
Outline - Two Parts

I. ***Seer*** [1]: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices, *ASPLOS '19, Cornell SAIL Group*

II. Put ***Seer*** Into A Broader Perspective - Performance Engineering for Cloud Microservices

[1] Gan, Yu, et al. "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices." *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2019.

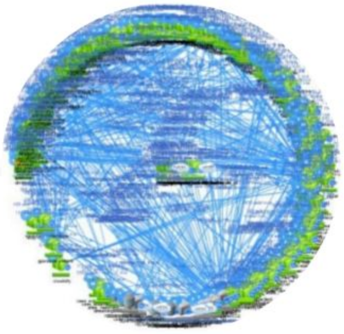
From Monolith to Microservices



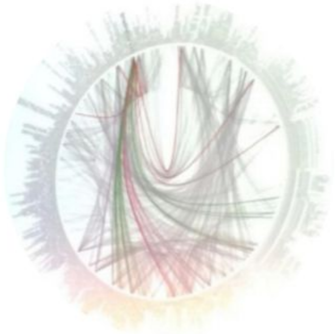
- All functionality in a single service
  - makes application evolution cumbersome and error prone
  - limits programming languages, tools, frameworks, etc.
- Fine-grained, single-concerned, loosely-coupled services
  - Modularity, Flexibility, Faster Dev, Elasticity...
  - Stricter QoS & Cascading QoS Violations & Unpredictable Performance -> A-posteriori Debugging Impractical

From Monolith to Microservices

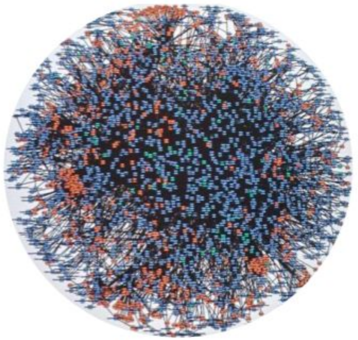
- ➢ Microservices are getting popular & it's important to do guarantee "service level agreement/objectives" for provided services @Amazon, Netflix
- ➔ Seer's Goal: proactive performance debugging for interactive microservices
  - ○ to anticipate & localize QoS violations (accuracy & completeness)
  - ○ to provide insights to improve microservices design and deployment



Netflix

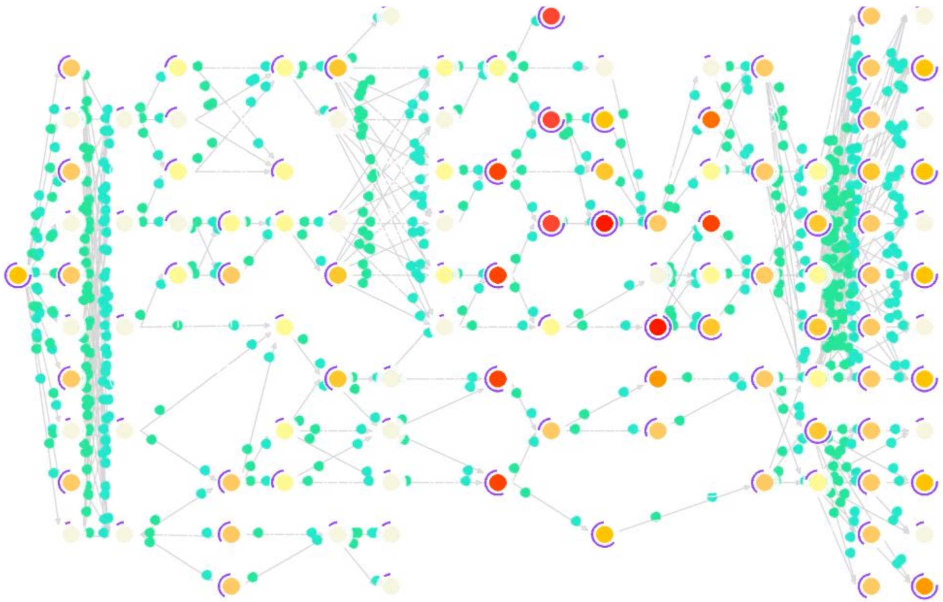


Twitter



Amazon

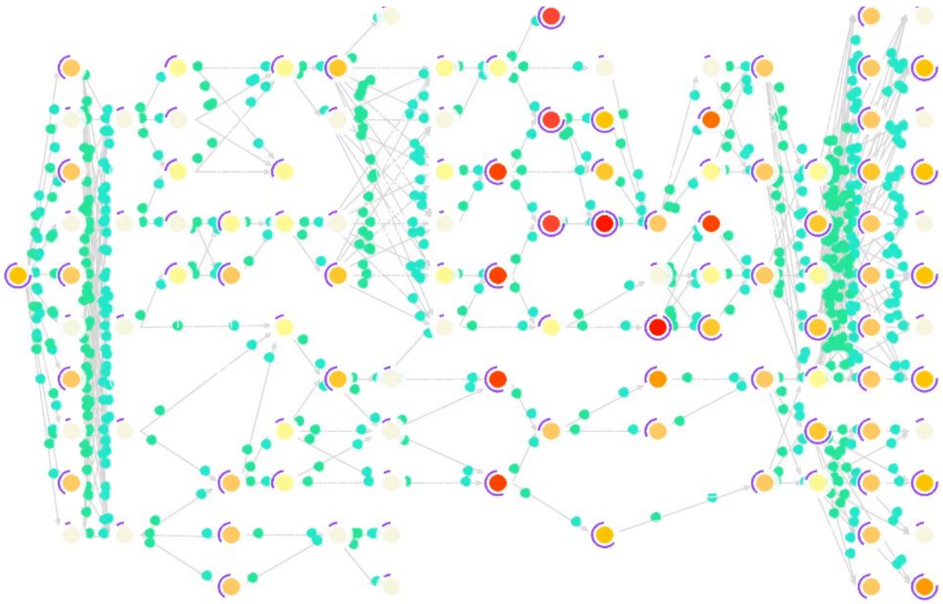
Perf-Debugging Challenges for Microservices (1)



- Complicated cluster management & performance debugging:
  - stricter QoS requirements
  - dependencies cause cascading QoS violations
  - difficult to isolate root cause of performance unpredictability

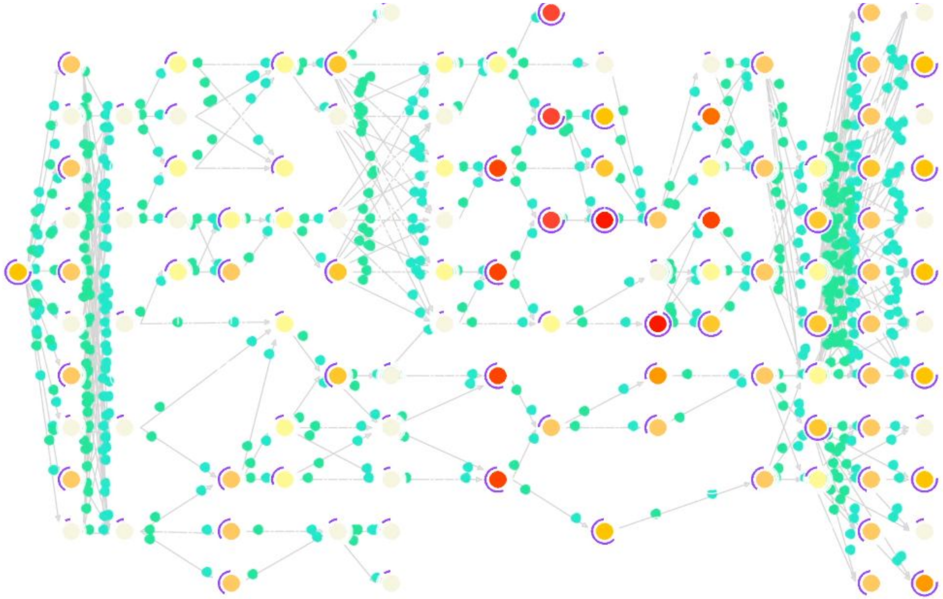
Demo: http://www.csl.cornell.edu/~delimitrou/2019.asplos.seer.demo_motivation.mp4

Perf-Debugging Challenges for Microservices (2)



- Empirical performance debugging ⇨ too slow ⇨ bottlenecks propagate
  - impractical by manual check or user feedback given the scale and complexity
  - long recovery time for performance

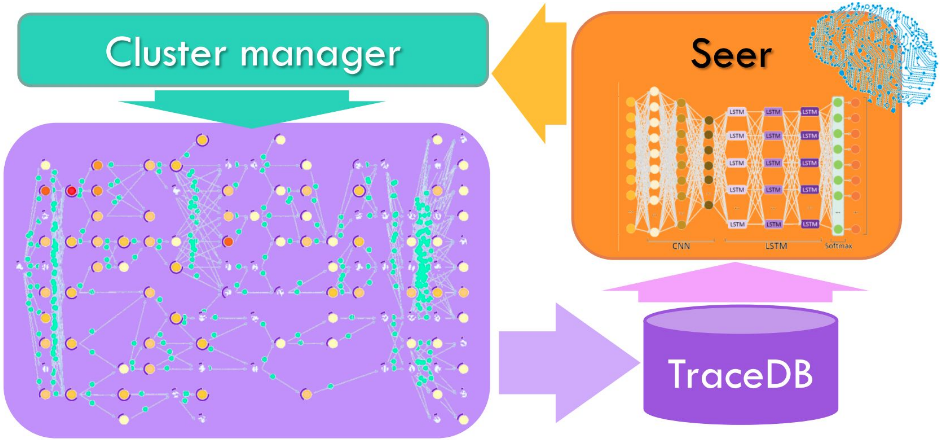
Perf-Debugging Challenges for Microservices (3)



- Hard to guarantee predictable performance
  - datacenter hardware heterogeneity
  - frequent server replacement & application update

Seer Overview

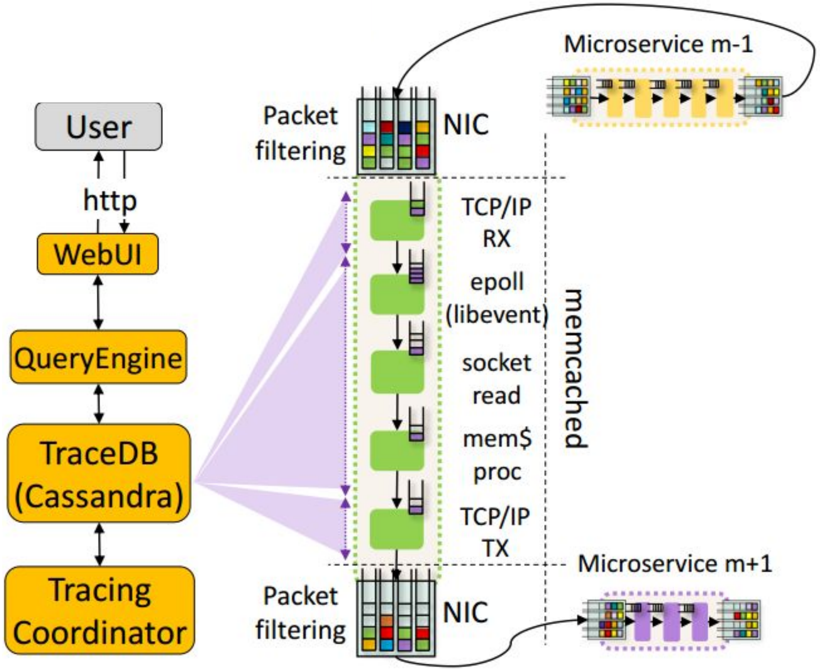
- Uses ML to identify the root cause of an upcoming QoS violation:
  - massive amount of distributed traces collected over time
  - uses targeted per-server hardware probes to determine the cause of the QoS violation



- Informs admin to take proactive action & prevent QoS violation
  - needs to predict 100s of msec -> a few sec in the future (not yet met)

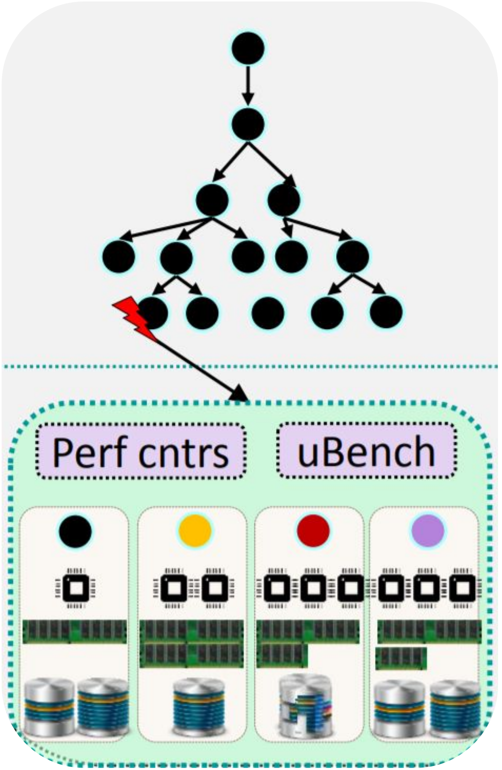
Two-level Tracing (1)

- Distributed RPC-level tracing
  - similar to Dapper, Zipkin, based on Apache Thrift timing interface
  - collects
    - per-microservice latencies
    - inter- and intra-microservice queue lengths
  - overhead: <0.15% in QPS, <0.1% in latency
  - traces are associated and aggregated in a Cassandra database
- Fine-grained instrumentation
  - distinguish network processing and application computation



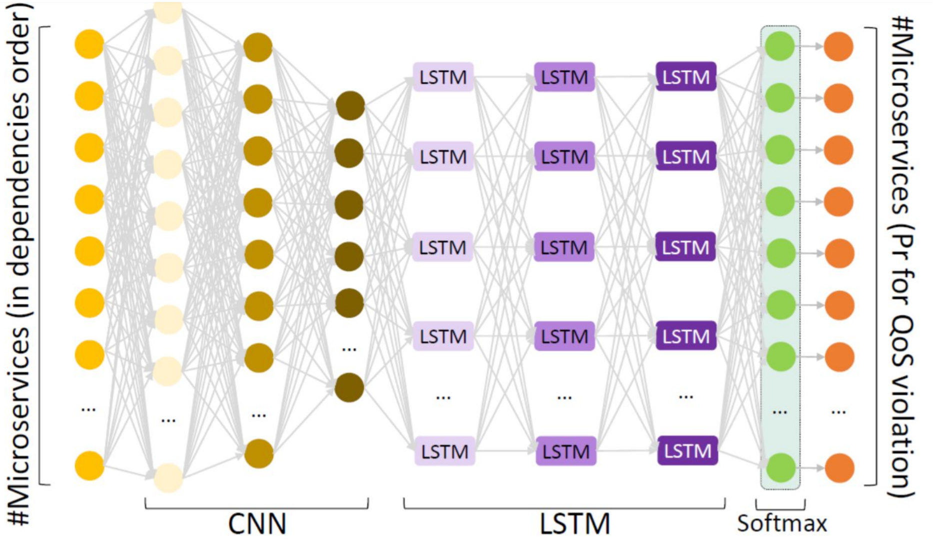
Two-level Tracing (2) - Hardware Monitoring

- Per-node (problematic microservices) hardware monitoring
  - Private cluster: performance counters & utilization monitors
    - CPU, memory capacity and bandwidth, network bandwidth, cache contention, storage I/O bandwidth
  - Public cluster: 10 tunable contentious microbenchmarks
    - targeting on different shared resources
    - each takes 10ms to complete



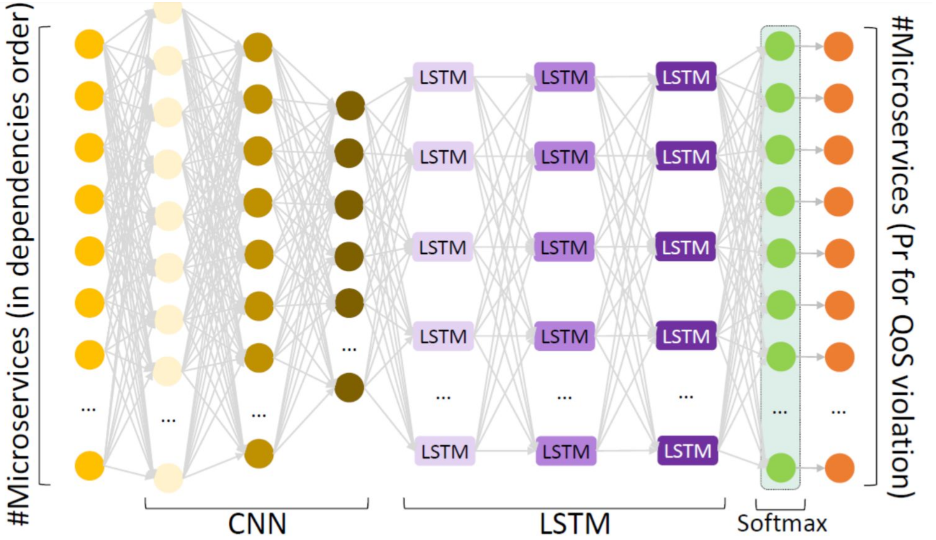
DL for Performance Debugging (1)

- Architecture-agnostic
- Adjust to intra-microservice changes over time
- High accuracy
- Good scalability
- Fast inference (within window of opportunity)



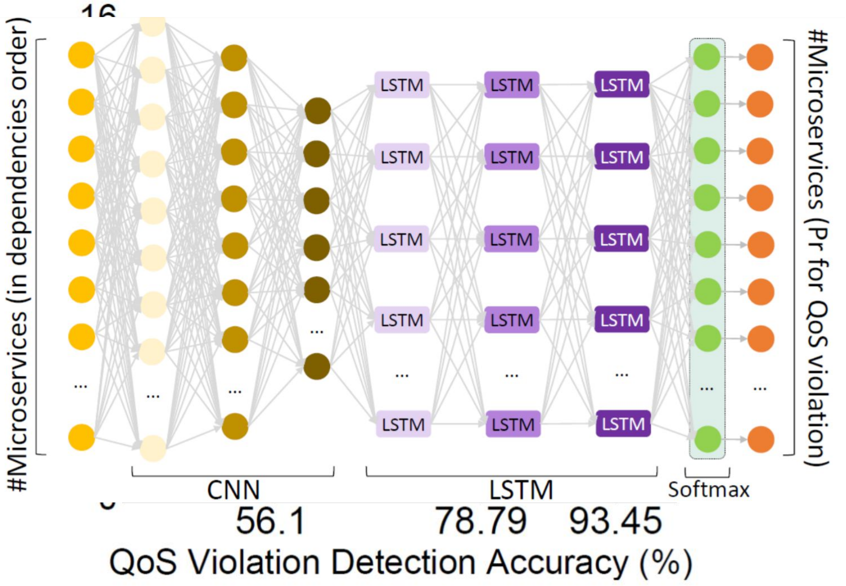
DL for Performance Debugging (2)

- Input signals:
  - container utilization ❌
  - latency ❌
  - queue length ✅
- Output signal:
  - probability that a microservice will initiate a QoS violation in the near future



DL for Performance Debugging (3)

- CNN: fast, but cannot predict near future
- LSTM (RNN): high accuracy, but affected by noisy, non-critical microservices
- Hybrid: high accuracy, w/o significantly higher overhead
  - CNN reduces the dimensionality & filters out non-critical microservices
  - LSTM+Softmax infer the probability of QoS violation

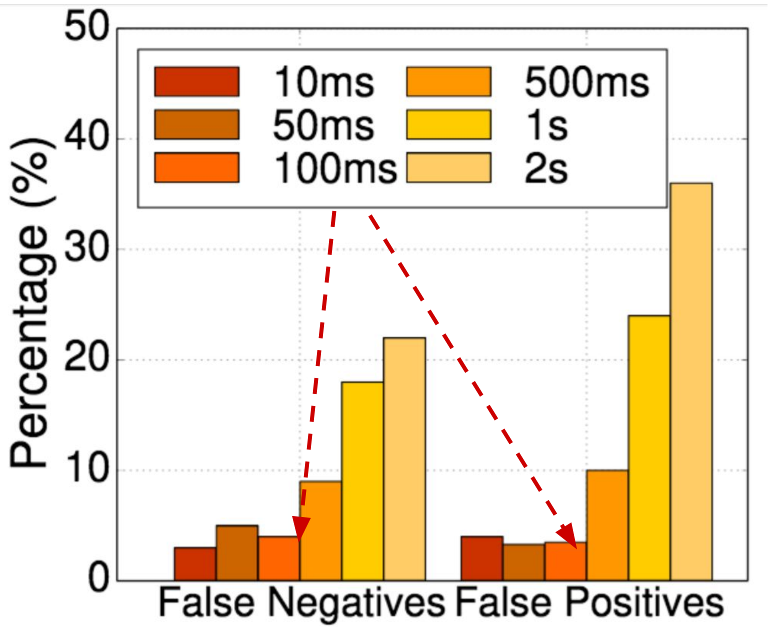


Methodology & Evaluation

- Training once: slow (hours - days)
    - across payloads, load distributions, request types
    - inject microbenchmarks to force controlled QoS violations -> annotated queue traces
    - weight/bias inference and optimization by SGD
- Incremental retraining in background (manually triggered)
    - more data: transfer learning-based approach
    - application/environment changes in a major way: retrain from scratch
- Inference: continuously streaming traces
    - 20-server dedicated heterogeneous cluster (different server configurations)
    - 10s of cores, >100GB RAM per server
- 4 end-to-end apps: Social Network, Media Service, E-commerce Site, eBank
    - 30-40 unique microservices each

Validation (1)

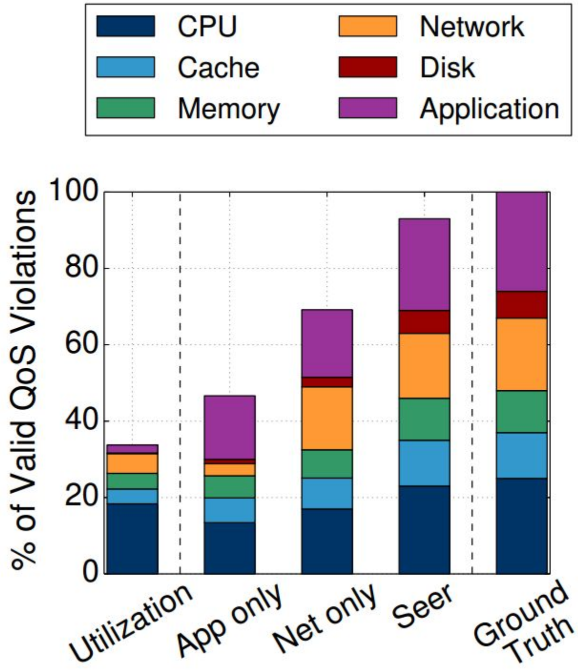
- Setting
  - 100GB tracing data (levels off afterwards)
  - 100ms tracing sampling interval (no benefit for more fine-grained sampling)
- 91% accuracy in signaling upcoming QoS violations
- 88% accuracy in attributing QoS violation to correct microservice



Prediction of QoS Violations in the Next 100ms

Validation (2)

- Setting
  - 100GB tracing data (levels off afterwards)
  - 100ms tracing sampling interval (no benefit for more fine-grained sampling)
- Seer can foresee 91% injected QoS violations (84/95 early enough to take action)
  - Utilization: threshold-based approach
  - App-only: limited version of Seer (only app queues)
  - Net-only: limited version of Seer (only net queues)
  - Ground-truth: injection campaign



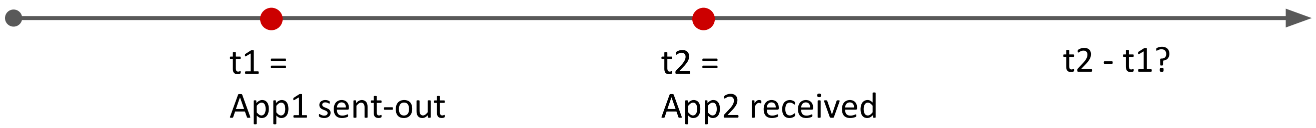
Prediction of QoS Violations in the Next 100ms

Avoiding QoS Violations

- Identify cause of QoS violation
  - private cluster: performance counters & utilization monitors (threshold-based)
  - public cluster: contentious microbenchmarks (check one by one)
- Adjust resource allocation
  - CPU/Memory/IO contention: resize containers
  - Cache contention: Intel cache allocation technology (CAT) for last level cache (LLC) partitioning
  - Network contention: Linux traffic control's hierarchical token bucket (HTB) queueing discipline in qdisc for bandwidth partitioning
- Application-level bugs
  - human intervention

Questions/Discussion

- Not considered: logic layer load-balancer, replicated instances for each individual micro-service
- Intensive Instrumentation + Expert knowledge on microservices
  - Collecting application & network queue lengths
  - Seer doesn't require domain knowledge about the dependencies (why not?)
- DL: predicting incoming workload patterns (randomness is not predictable)
  - As expected, Seer doesn't foresee random violations: network switch failure, random load spikes, which is usually the case in practice.
- Timing Synchronization on Tracing Data



So Far…

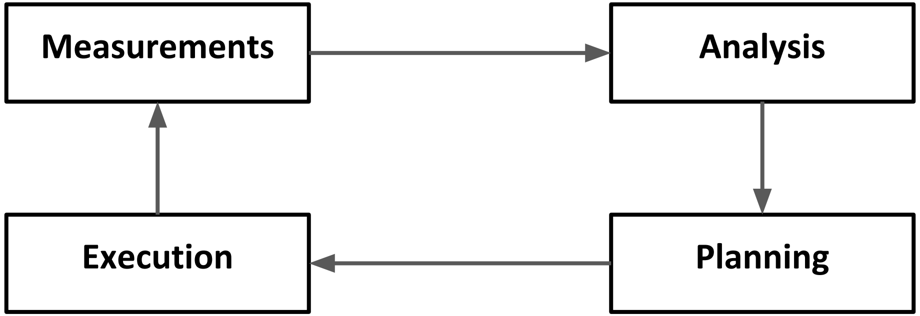
- Microservices become increasingly popular
- Traditional performance debugging techniques do not scale and introduce long recovery times

Seer's Contribution

- Seer leverages DL to anticipate QoS violations & find their root causes
  - 91% detection accuracy, avoids 86% of QoS violations
- [?] Seer provides insights on how to better design and deploy complex microservices
- [?] Seer provides practical solutions for systems whose scale make previous empirical solutions impractical

Performance Engineering for Cloud Microservices

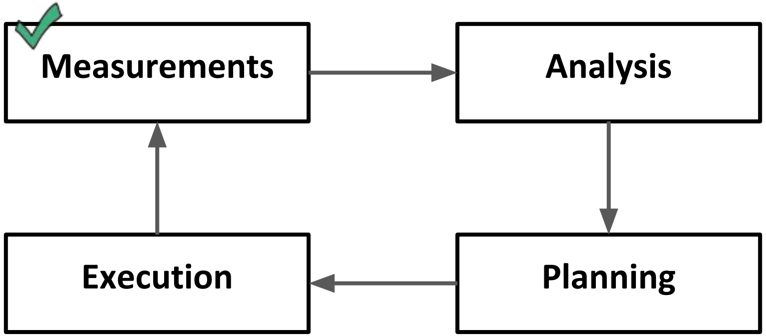
- Why Cloud Computing?
  - Elasticity: dynamically acquire and release resources according to the need;
- [*Classical Problem + New Setting*] To achieve efficiency, it's vital to **automatically and timely** provision and deprovision the **right type and amount** of cloud resources to cater to **dynamic workloads** without breaking the **QoS/SLA**s.
- **MAPE** Loop



MAPE Loop (1)

Seer ✓

- Performance Indicator
- Monitoring Interval



- ✓ Prediction of QoS Violation/Performance/Workload
- ✓ Adaptivity to Changes
- Oscillation Mitigation

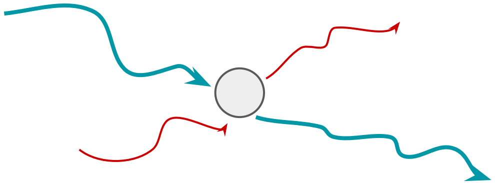
- ✓ Execute Actions
- Cloud Providers' APIs

- Resource Estimation
- ✓ Possible Actions
- Cost Model

MAPE Loop (2) - **Single Application**

- Resource Estimation (for given workload)
  - Threshold/Rule-based
  - Fuzzy Inference
  - Application Profiling
  - Analytical Modeling (Queueing Theory)
  - Machine Learning (CNN, Reinforcement Learning)
- Workload Prediction
  - Time-series Analysis: linear regression, autoregressive models (AR), moving average (MA), ARMA, Kalman filter, neural networks
- Other Approaches
  - Resource Usage Prediction: linear regression, neural networks, ARMA, etc.

MAPE Loop (3) - **Microservices**



- Divide and Conquer
  - break overall SLA into SLA of each service
  - satisfy individual SLAs to meet the overall SLA
  - problem: some services serve multiple execution paths & hard to know exact SLA breakdowns

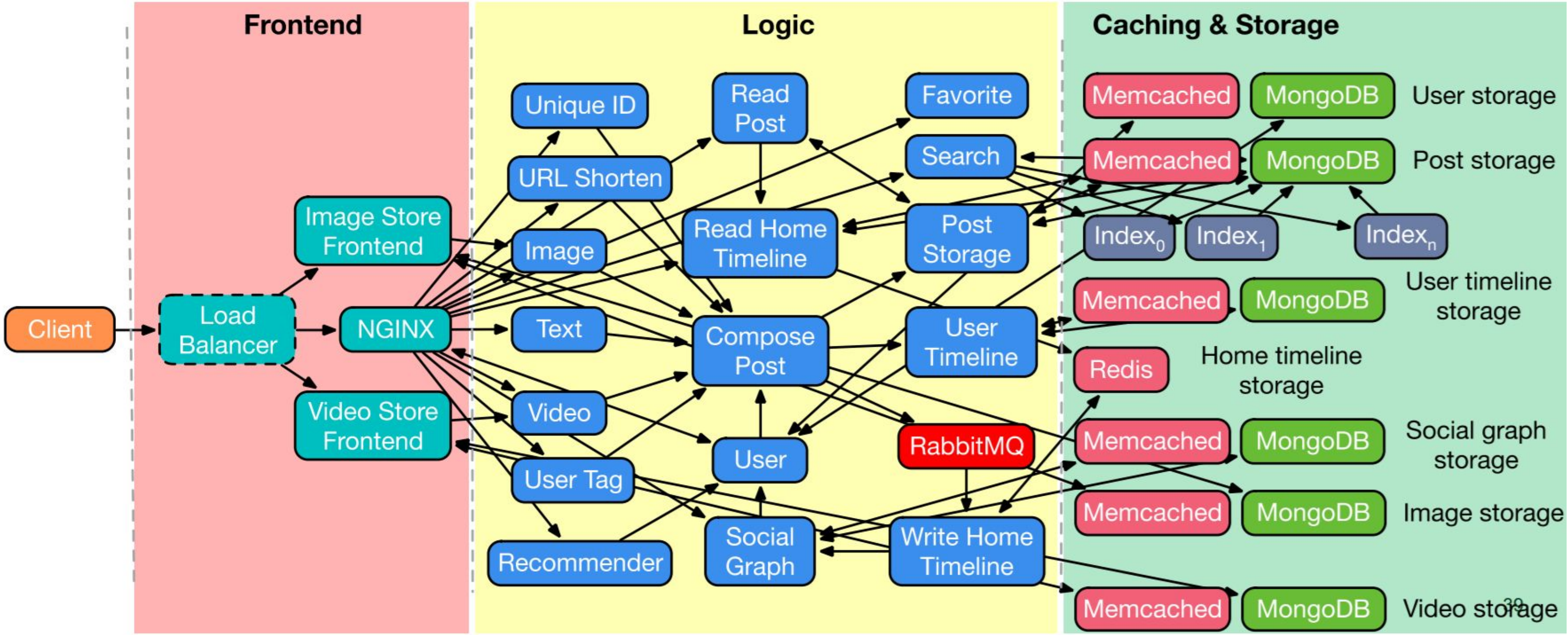
- Bottom-up Approach
  - what-if analysis: each service estimates the change of performance if adding or removing one instance
  - aggregate to choose the operations that optimizes the performance
- Performance Violation Prediction + Per-instance Monitoring (**Seer**)

References

- Seer, ASPLOS 2019, http://www.csl.cornell.edu/~delimitrou/papers/2019.asplos.seer.pdf
- µqSim: Enabling Accurate and Scalable Simulation for Interactive Microservices, http://www.csl.cornell.edu/~delimitrou/papers/2019.ispass.qsim.pdf
- Seer Demo: https://www.youtube.com/watch?v=Mf_C2xCpBdc
- Auto-scaling Web Applications in Clouds: https://arxiv.org/abs/1609.09224
- A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments: https://link.springer.com/content/pdf/10.1007/s10723-014-9314-7.pdf

Thank you!

Appendix - Social Networks



Appendix - Survey on Auto-scaling