FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-oriented Microservices


DEPEND Research Group
University of Illinois at Urbana-Champaign

* Presenter
From Monolithic to Microservices

- Microservice architecture growing in popularity
- A set of loosely-coupled, self-concerned “micro” services
  - Scalability, fault isolation, flexibility, etc.
- Scale and complexity are increasing
  - Increasing in scale, e.g. 700+ (Netflix in ’17), 1000+ (Uber in ’19)
- Performance guarded by service level objectives (SLOs)
  - Violation leads to financial loss (100ms increase converted to $0.7 billion loss in Amazon sales (Q4 ’18)
Performance Predictability in Microservices is Hard

**Challenge #1: Difficulty** in isolating root causes of SLO violations
- Complex inter-microservice dependencies cascading SLO violations

**Challenge #2: Inability** in capturing shared-resource contention at a lower-level
- Interference over shared resources (e.g. LLC, memory bandwidth, network devices)

**Challenge #3: Difficulty** in taking the right action to mitigate SLO violations
- High fidelity performance models/scheduling heuristics -> significant human-effort and training
- Frequent service updates/migrations -> recurring effort for model reconstruction and re-training
FIRM As The Cure

- Two-level machine learning based SLO violation mitigation framework
  - **Challenge #1** – Detection and localization of SLO violations to individual microservices
  - **Challenge #2 & #3** – Estimation of resources in contention and dynamic resource reprovision
- **Benefits**: Improved interpretability and less training time

- Designed, developed, and deployed in a 15-node Kubernetes cluster
- Outperforms Kubernetes autoscaling by **up to 16x** in reducing SLO violations
**Insight 1: Dynamic Behavior of Critical Paths**

- Critical path defines the longest path in execution.
- Detection of critical paths helps reveal the bottleneck of performance.
- Critical path is not static, but dynamically changing based on the performance of individual service instances:
  - Different type of underlying shared-resource contention.
  - Different degree of sensitivity to the same type of interference.
- It’s important to capture the changes at runtime, and make runtime decision.
**Insight 2: Significance of Latency Variability**

- Microservices with larger latency are not necessarily the root causes of SLO violations.
- Processing time with higher variance makes it harder to obtain low tail latency.
- Variability represents opportunities for reducing latency.

![Graph showing CDF for individual and total latency with annotations](image)

**Social Network – Composing Post Request**
State Inference (1)

- Real-time observability on request execution provided by end-to-end distributed tracing
- Auto-labeled training data driven by the performance anomaly injection
State Inference (3)

- Real-time observability on request execution provided by end-to-end distributed tracing
- Auto-labeled training data driven by the performance anomaly injection
- SLO violation detection and narrow down via critical path analysis
- SVM-based critical component localization
  - Given individual latency vector $T_i$, and end-to-end latency vector $T_{CP}$
  - Relative importance defined as the Pearson correlation coefficient between $T_i$ and $T_{CP}$
  - Congestion intensity defined as 99-th percentile value divided by median value of $T_i$
Insight 3: No Common Mitigation Policy for All

• SLO violation mitigation policies vary with applications, user loads, and the types of resource in contention

• Designing optimal resource provisioning strategy is intractable, just like scheduling problems
  • Modeling complexity: Tetris [SIGCOMM ’14], Jokey [EuroSys ’12]
  • Placement constraints: TetriSched [EuroSys ’16], device placement [NIPS ’17]
  • Data locality: Delayed scheduling [EuroSys ’10], SWAG [SoCC ’15]
  • ...
Why not human-driven performance engineering?

• No “one-size-fits-all” solution for the online decision problem
  • Best algorithm depends on specific workload and system

• Human-driven performance engineering
  • Assume a simple system model
  • Produce some clever heuristics
  • Painstakingly test & tune the heuristics in practice
  • Redo the above steps

• RL-based SLO violation mitigation
  • Assume a random scheduling policy
  • Perceive states and receive rewards
  • Optimize the policy based on the rewards
  • Loop continues until convergence

• Is there a way to work around human-generated heuristics? Yes
SLO Violation Mitigation (1)

- Observability improved through online distributed tracing
- Auto-labeled training data and RL online learning driven by the performance anomaly injector
- SLO violation detection and localization via critical path analysis
- SVM-based critical component extraction
- SLO violation mitigation based on reinforcement learning
  - Identifies low-level resource contention
  - Estimates the right amount to reprovision
SLO Violation Mitigation (2)

- An RL-based resource estimation agent that learns to make provisioning decisions directly from experience
- Optimizes objectives end-to-end:
  - Minimize SLO violation
  - Maximize resource utilization efficiency

\[
 r(t) = \alpha \cdot SM_t \cdot |\mathcal{R}| + (1 - \alpha) \cdot \sum_{i}^{\mathcal{R}} \frac{RU_i}{RL_i}
\]

Mitigate SLO Violation Fast
Avoid Over-provisioning

\[
 SM_t = \frac{Latency_{SLO}}{Latency_t} \quad \text{(SLO maintenance)}
\]

Resource utilization of \(i\) at time \(t\)
Resource limit of \(i\) at time \(t\)
Multilevel ML Training

- Reinforcement Learning Training
- FIRM’s RL Agent
- K8S Cluster
- Anomaly Injector
- FIRM’s SVM Model

Workload Generators

Generate experience data

Feature (X)

Label (y)

Labeling
Multilevel ML Training

- K8S Cluster
- Anomaly Injector
- Workload Generators
- FIRM’s SVM Model
- Feature (X)
- Label (y)

Graphs:
- Total Reward vs. Episode
- Mitigation Time vs. Episode

- One-for-All
- One-for-Each
- Transferred

- K8S Autoscaling
- AIMD

- 2x
- 9x

- Reinforcement Learning Training
  - Generate experience data
  - Anomaly Injector
  - Labeling
Evaluation

- Implemented and deployed FIRM on a Kubernetes cluster of 15 physical nodes
- Running microservices benchmarks from DeathStarBench [1] and TrainTickets [2] driven by open-loop workload generators
- Training and experiments driven by performance anomaly injection
- Comparison targets include Kubernetes autoscaling [3] and an additive increase multiplicative decrease (AIMD)-based method [4]

---


Results

- Reduces the **SLO violation mitigation time** by up to 9× compared with AIMD
- Reduces the average **tail latencies** by up to 6-11×
- Reduces the overall average **requested CPU limit** by 29-62%
- Reduces the number of **dropped/timed out requests** by up to 8x
Conclusion

• FIRM uses SVM-based critical component extraction to localize at runtime root cause microservice instances for SLO violations
• FIRM uses RL to generate workload-specific mitigation policies, optimized to estimate resources in contention and provide re-provision actions
• FIRM leverages the two-level ML model structure to improve interpretability and save training time
• FIRM outperforms Kubernetes auto-scalers and AIMD-based methods
Thank you!

Check out the full paper for more details!
(haoranq4@illinois.edu)