Automating Performance Tuning with Machine Learning

USENIX SRECon 21
Stefano Doni (Akamas)
Agenda

1. Why SREs should care about system configurations
2. A new approach: ML-driven performance tuning
3. Real-world example: optimize Kubernetes and JVM
4. Conclusions

Stefano Doni
CTO at Akamas
15 years in performance engineering
2015 CMG Best Paper Award Winner
Why SREs should care about system configurations
SREs care about efficiency and performance

“an SRE team is responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of their service(s)”

The core SRE tenets include:
- Pursuing maximum change velocity without violating SLOs
- Demand Forecasting and Capacity Planning
- Efficiency and performance

https://sre.google/books
Tuning system configuration matters...

**Performance and efficiency**
- Higher application performance and lower infrastructure cost

**...and service availability**
- Higher transaction throughput and improved service resilience

© 2021 Akamas • All Rights Reserved • Confidential
... but it is getting harder and harder

**Configuration Explosion**

Properly configuring the IT stack requires analyzing thousands of configurations.

**Unpredictable Effects**

Effect of changes can be counterintuitive + default values not always appropriate.

**Faster Deployments**

Acceleration of release pace makes manual approach infeasible/useless.
A new approach: ML-driven performance tuning
Key requirements for a new approach

- **Full-Stack**: Optimize multiple technologies and layers at the same time.
- **Smart Exploration**: Explore huge space of configurations in a time and cost-effective way.
- **Goal-oriented**: Define tailored goals and constraints driving the optimization.
- **Fully Automated**: Execute the entire optimization process in a fully automated way.
ML techniques for smart exploration

**Model Based**
- Queuing Networks
- Petri Networks
- Linear Programming

**Simulation Based**
- Random Forests
- Statistical Machine Learning

**Test Based**
- Random Search
- Reinforcement Learning
- Parzen Trees
ML enables automated performance tuning...

1. Apply Configuration
2. Apply Workload
3. Collect KPIs
4. Score vs Goal (RL reward)
5. Reinforcement Learning Optimization

System to be Optimized (RL Environment)

Configuration mgmt tools
- Adjust tunable parameters of the system (RL Action)

Load Testing tools
- Test the new parameter configuration under load

Monitoring tools
- Measure performance KPIs from monitoring tools

Tools (OS / HW, Container / Pod, Application, Framework (DB), Runtime (JVM))
... and a new performance tuning process

1. Apply Configuration
2. Apply Workload
3. Collect KPIs
4. Score vs Goal
5. Reinforcement Learning Optimization

- Optimization goal
- Constraints (SLOs)
- Load scenarios

SRE

optimal configuration

© 2021 Akamas • All Rights Reserved • Confidential
Real world example: optimize Kubernetes and JVM
The target system: Online Boutique

- **Cloud-native application** by Google made of **10 microservices**
- Realistic sample web-based **e-commerce service**
- Features a **modern software stack** (Go, Node.js, Java, Python, Redis)
- Includes a Load Generator (Locust) to inject **realistic workloads**

https://github.com/GoogleCloudPlatform/microservices-demo
Use Case: optimizing cost of K8s microservices while ensuring reliability

Challenge for SRE

How to provision the optimal resources to your application made of several Kubernetes microservices, so that you can trust the overall service

➔ will sustain the expected target load
➔ while matching the defined Service-Level Objectives (SLOs)
➔ at the minimum cost
➔ while minimizing the operational effort
➔ and matching delivery milestones
The reference architecture

Automated Workflow

Load Generator

Configuration Mgmt

Frontend

Payment
Currency
Recommend
EMailing
Ad

Shipping
Product Catalog
Check-out
Cart
Redis Cart

Monitoring

22 TUNABLE PARAMETERS (CPU & Memory limits)

10 MICROSERVICES

275 MEASURED KPIs
The optimization goals & constraints

**GOAL:**

*MAXIMIZE* \( \frac{\text{frontend\_boutique\_istio\_incoming\_success\_transactions}}{\text{application\_cost}} \)

**CONSTRAINTS:**

*loadgenerator\_locust\_locust\_fail\_ratio \leq 2\% AND* \( \text{frontend\_boutique\_istio\_incoming\_response\_time\_90pct} \leq 500\text{ms} \)
Best configuration found by ML in 24H improves cost efficiency by 77%

~35 iterations
~24 hours elapsed

Baseline configuration
Perf/Cost: 0.29 TPS/$/mo

Best configuration
Perf/Cost: 0.52 TPS/$/mo

+77%
Best config: optimal resources assigned to microservices

- decreased CPU limits set for almost all containers
- increased CPU assigned to 2 microservices
- all these changes to achieve max cost efficiency and match SLOs
Best config: higher performance & efficiency for the overall service

Baseline vs Best: Service throughput

Baseline vs Best: Service p90 response time

+19% TPS

-60% Response Time
Use Case: maximizing service performance & efficiency with JVM tuning

Challenge for SRE

How to ensure a reliable product launch, by properly configuring JVM options, so that you can trust the overall service

- will sustain the expected target load
- while matching the defined Service-Level Objectives (SLO)
- at the minimum cost
- while minimizing the operational effort
- and staying aligned product launch milestones
The reference architecture

- **Automated Workflow**
  - Load Generator

- **Frontend**
  - Configuration Mgmt
  - Payment
  - Shipping
  - Currency
  - Product Catalog
  - Recommend
  - Check-out
  - EMailing
  - Cart
  - Ad
  - Redis Cart

- **Monitoring**

- **32 TUNABLE PARAMETERS (JVM options)**

- **10 MICROSERVICES**

- **275 MEASURED KPIs**
The optimization goals & constraints

**GOAL:**
MAXIMIZE \( \text{ad.istio_incoming_success_transactions} \)

**CONSTRAINTS:**
\( \text{ad.transaction_response_time} \leq 100\text{ms} \)
Best config: +28% throughput, and meeting SLOs

Baseline configuration
Peak Throughput matching SLO: 95 TPS
SLO breaking at 100ms

Best configuration
+28%
Peak Throughput matching SLO: 74 TPS
Best config: optimal JVM options

8
TOP IMPACT PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Baseline</th>
<th>Best</th>
<th>Relevance</th>
</tr>
</thead>
<tbody>
<tr>
<td>jvm_newSize</td>
<td>300 MB</td>
<td>550 MB (+83.3%)</td>
<td></td>
</tr>
<tr>
<td>jvm_GCTimeRatio</td>
<td>90</td>
<td>100 (+1%)</td>
<td></td>
</tr>
<tr>
<td>jvm.concurrentGCThreads</td>
<td>8 threads</td>
<td>1 threads (+47.7%)</td>
<td></td>
</tr>
<tr>
<td>jvm_gcType</td>
<td>parallel</td>
<td>n.s.</td>
<td></td>
</tr>
<tr>
<td>jvm_maxHeapSize</td>
<td>256 MB</td>
<td>901 MB (+252%)</td>
<td></td>
</tr>
<tr>
<td>jvm_maxTenuringThreshold</td>
<td>15</td>
<td>6 (+60%)</td>
<td></td>
</tr>
<tr>
<td>jvm_parallelGCThreads</td>
<td>8 threads</td>
<td>3 threads (+92.3%)</td>
<td></td>
</tr>
<tr>
<td>jvm_survivorRatio</td>
<td>8</td>
<td>100 (+1.15%)</td>
<td></td>
</tr>
</tbody>
</table>

- increased max heap memory
- changed Garbage Collector type
- decreased number of Garbage Collector threads
- adjusted heap regions & object aging thresholds
Conclusions
Key takeaways

Tuning modern applications for increasing their efficiency, performance and reliability is a complex problem that represent a relevant toil for SRE teams.

A new approach leveraging fully-automated ML-based optimization enables SRE teams to ensure applications will have higher performance & reliability.

Leveraging this new ML-based optimization approach, SRE teams can reduce the operational toil and stay aligned to release milestones.
Contacts

Italy HQ
Via Schiaffino 11
Milan, 20158
+39-02-4951-7001

USA East
211 Congress Street
Boston, MA 02110
+1-617-936-0212

USA West
12655 W. Jefferson Blvd
Los Angeles, CA 90066
+1-323-524-0524

Singapore
5 Temasek Blvd
Singapore 038985

LinkedIn
@akamaslabs

Twitter
@AkamasLabs

Email
info@akamas.io
BACKUP SLIDES