Leveraging Kubernetes Source Code for Performance Simulation

13th Symposium on Software Performance 2022
Session 4: Performance from Cloud to Edge

Martin Straesser, Patrick Haas, Samuel Kounev

09.11.2022

https://se.informatik.uni-wuerzburg.de
Introduction
Container orchestration automates the deployment, management, scaling, and networking of containers. […]

Container orchestration is used to automate and manage tasks such as:

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling […]
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

Source: Red Hat Inc.
https://www.redhat.com/en/topics/containers/what-is-container-orchestration
Introduction

Why container orchestration (CO) in software performance research?

- CO mechanisms have implications on performance of managed applications [1, 2, 3]
- Container orchestrators themselves are distributed applications with interesting performance characteristics [4, 5]
- All of the mentioned tasks are non-trivial and be assessed using different approaches
- Container orchestration is a high-valued task in production environments as it is (co-) responsible for availability, quality of service, operating costs, resilience, security etc. of cloud applications

Challenges

- Holistic view on container orchestration and interdependencies between CO tasks
- Modeling is hard because of system complexity and continuous updates
Problem

- Goal
  - Test different container orchestration policies

- Obstacles
  - Needs complex technical setup
  - Needs suitable load generation
  - Limited reproducibility
  - Bound by costs

- Proposed solution: Performance simulation with integrated container orchestration functions using their original implementation
  - Save costs for experimental evaluation
  - Produce simulation results of high quality
Simulating Container Orchestration

- We build on the state-of-the-art microservice simulation MiSim [6]

MiSim Features

- Discrete Event Simulation
- Specialized on microservices
- Implements CPU performance model
- Implements several resilience mechanisms
- Supports fault injections and dynamic workloads

Orchestration Extension

- Supports all features of MiSim
- Adds model of nodes and containers
- Implements several CO mechanisms, e.g. health monitoring, scheduling, ...
- Allows to include original Kubernetes configuration files for deployments, pods, autoscalers etc.
- Provides interfaces to use original kube-scheduler and cluster-autoscaler in simulation
Framework Overview

MiSim Config Files → MiSim → Simulation Output, Graphs, …
Leveraging Kubernetes Source Code for Performance Simulation

Martin Straesser, Patrick Haas, Samuel Kounev
Leveraging Kubernetes Source Code for Performance Simulation

Martin Straesser, Patrick Haas, Samuel Kounev

Framework Overview

MiSim

K8S Config Files

apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: frontend-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50

Simulation Output, Graphs, …

Additional Statistics
Leveraging Kubernetes Source Code for Performance Simulation

Martin Straesser, Patrick Haas, Samuel Kounev

Framework Overview

MiSim Config Files

K8S Config Files

Orchestration Extension

Simulation Output, Graphs, …

Additional Statistics

MiSim

Kubernetes Adapter

kube-scheduler

Scheduler Config

cluster-autoscaler
How It Works

- **Question 1:** Kubernetes components work normally in real-time with real cluster resources, how does this fit a discrete event simulation?

- **Answer**
  - By having a closer look at Kubernetes internal architecture, we note that communication is realized with event-based HTTP watch streams, which are emitted by the kube-apiserver.
  - Our Kubernetes adapter implements parts of the kube-apiserver, it basically translates MiSim events to Kubernetes events and vice versa.

- **Question 2:** What about the performance overhead?

- **Answer**
  - Performance overhead is around 10% in terms of simulation runtime, no significantly more resource usage.
Leveraging Kubernetes Source Code for Performance Simulation

**kube-scheduler - Evaluation Setup**

With Flannel Pod Requests

- **CPU**: 0.1
- **MEMORY**: 50Mi

**Simulation**

With Flannel Pod Requests

- **CPU**: 100
- **MEMORY**: 50Mi

**Kubernetes Cluster**

- **Small**: CPU 1, MEMORY 1938412Ki
- **Medium**: CPU 2, MEMORY 3936632Ki
- **Large**: CPU 4, MEMORY 8064996Ki

- **Small**: CPU 1000, MEMORY 1938412Ki
- **Medium**: CPU 2000, MEMORY 3936632Ki
- **Large**: CPU 4000, MEMORY 8064996Ki
Leveraging Kubernetes Source Code for Performance Simulation

**kube-scheduler - Evaluation**

- **Scheduling Policy**: NodeResourceBalancedAllocation

<table>
<thead>
<tr>
<th>Desired</th>
<th>K8s-Cluster</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node</td>
<td>Small</td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>large</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>large</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>medium</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>large</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>large</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>medium</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>small</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>N/A</td>
<td>1</td>
</tr>
</tbody>
</table>
## Scheduling Policy: NodeResourcesFit (CPU) – MostAllocated

<table>
<thead>
<tr>
<th>Desired</th>
<th>K8s-Cluster</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node</td>
<td>Small</td>
</tr>
<tr>
<td>0</td>
<td>N/A</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>small</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>medium</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>N/A</td>
<td>1</td>
</tr>
</tbody>
</table>
kube-scheduler - Evaluation

- Integrated kube-scheduler capable of reflecting different scheduling strategies in simulation
- We were even able to reproduce an active GitHub issue of the kube-scheduler
- However, not all custom scheduling plugins can be simulated out of the box (e.g. when nodes are grouped in zones and you want to have some “zone-aware” scheduling)
  - Solution: Provide “black-box” Kubernetes configuration files (e.g. node descriptions)
  - Simulation can not directly understand them but will forward the information to the kube-scheduler
Next step forward – cluster-autoscaler

1. Deploy new container
2. New desired container
3. No resources
4. No resources and container pending
5. Add new node
6. New desired container
7. Place container
8. Placement and Node
Conclusion

Problem
• Container orchestrators fulfill many performance-relevant tasks
• Modeling hard, experimental evaluation expensive

Idea
• Performance simulation with integrated container orchestration functions using their original implementation

Benefits
• Simulation with increased accuracy and more use cases
• Analysis of container orchestration policies
References


