上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Dynamically Adjusting Scale of a Kubernetes Cluster Under QoS Guarantee

Presenter: Li Lu

Qiang Wu, Jiadi Yu, Li Lu, Shiyou Qian, Guangtao Xue
Shanghai Jiao Tong University

Cluster Autoscaler for Kubernetes

- **Huge electricity consumption**
  - **Data centers consume approximately 1.12% of all electricity worldwide**
  - **A half of the operational expenses within a data center are consumed by the electricity cost**
- **Billing mechanism**
  - **Many cloud providers, such as Amazon, gradually support resource provisioning and billing in second manner**
- **Low cluster resource utilization**
  - **Cluster is generally designed to handle peak loads**
  - **During ordinary times, the load of a server is less than 50% of peak and the CPU utilization of a server rarely goes beyond 40%**

Our work

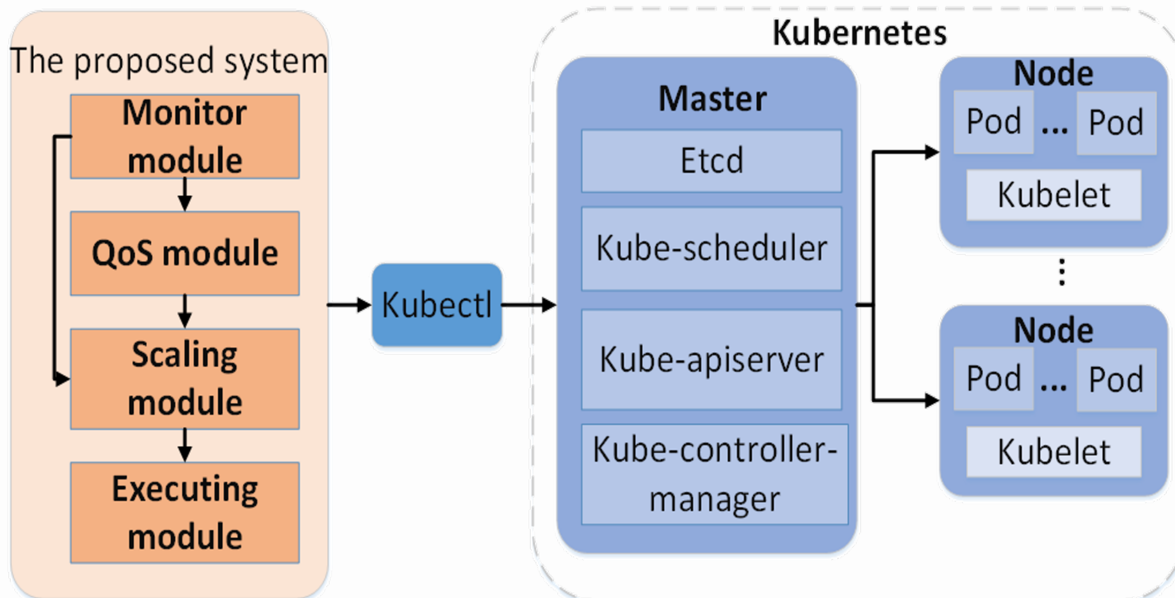
- Target to widely-deployed web applications
- Find out a threshold of resource utilization
  - Guarantee QoS in a Kubernetes cluster
  - Determine the time when to scale up the cluster
- Design a system to scale up or down the cluster
  - Guarantee quality of service
  - Improve the cluster resource utilization

Outline

- **System design**
- Evaluation
- Conclusion

Overview

- **Our system adopts a Monitor-Analyze-Plan-Execute (MAPE) model, include four modules:**
  - **Monitor module**
  - **QoS module**
  - **Scaling module**
  - **Executing module**



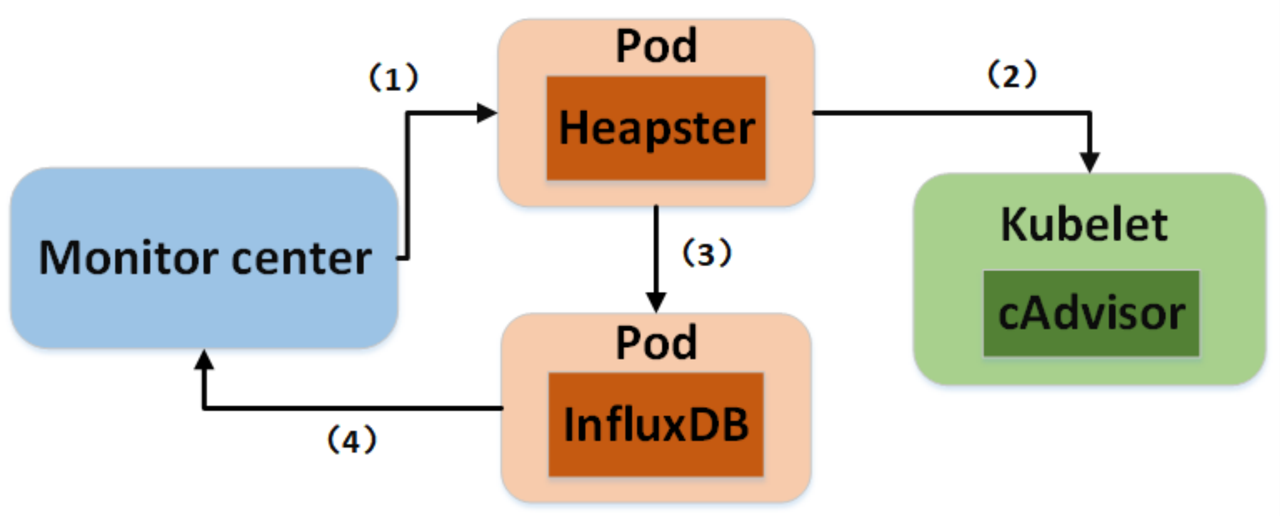
Monitor Module

- **Goal:**

  Monitor module is used to monitor CPU utilization of a whole Kubernetes cluster.

- **Workflow:**

  Monitor center --> Heapster --> cAdvisor --> Heapster --> InfluxDB --> monitor center

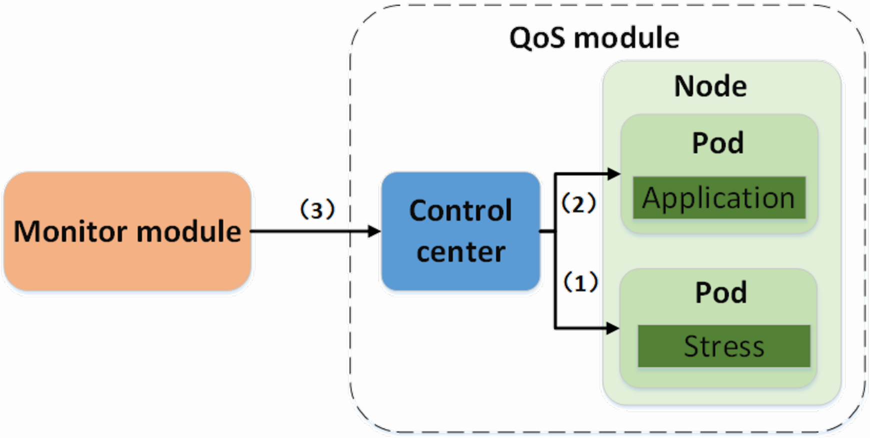


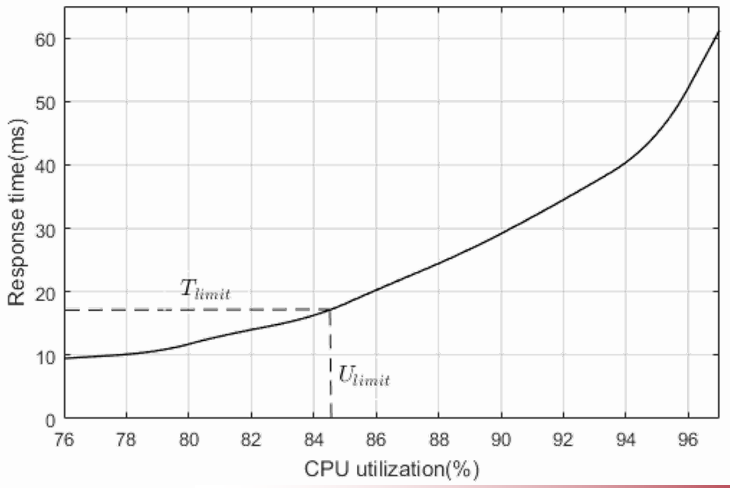
QoS Module

- **Initialized Parts**
  - Run once to obtain the relationship between QoS and CPU utilization
- **Goal:**
  - Obtain a proper threshold of CPU utilization
  - Guarantee quality of service
- **Metrics of QoS: response time**

Relationship between QoS and CPU

- **Workflow:**
  - **Step1: The control center sends a HTTP request to the application, and then receives the response to calculate the response time.**
  - **Step2: The control center gets CPU utilization from monitor module.**
  - **Step3: The control center changes CPU utilization of the server, and then rerun step1;**
- **Thus, we get the relationship:**





Threshold of CPU utilization

- **The upper bound of response time:**

$$T_{limit} = \alpha \times T_{normal}$$

T_{normal} is the response time whose relative CPU utilization is 40%

α is determined by users to meet their requirements

- **Thus, we get the threshold of CPU utilization $U_{threshold}$:**

$$U_{threshold} = \begin{cases} 90\% & U_{limit} \geq 90\% \\ U_{limit} & U_{limit} < 90\% \end{cases}$$

U_{limit} is the CPU utilization corresponding to T_{limit}

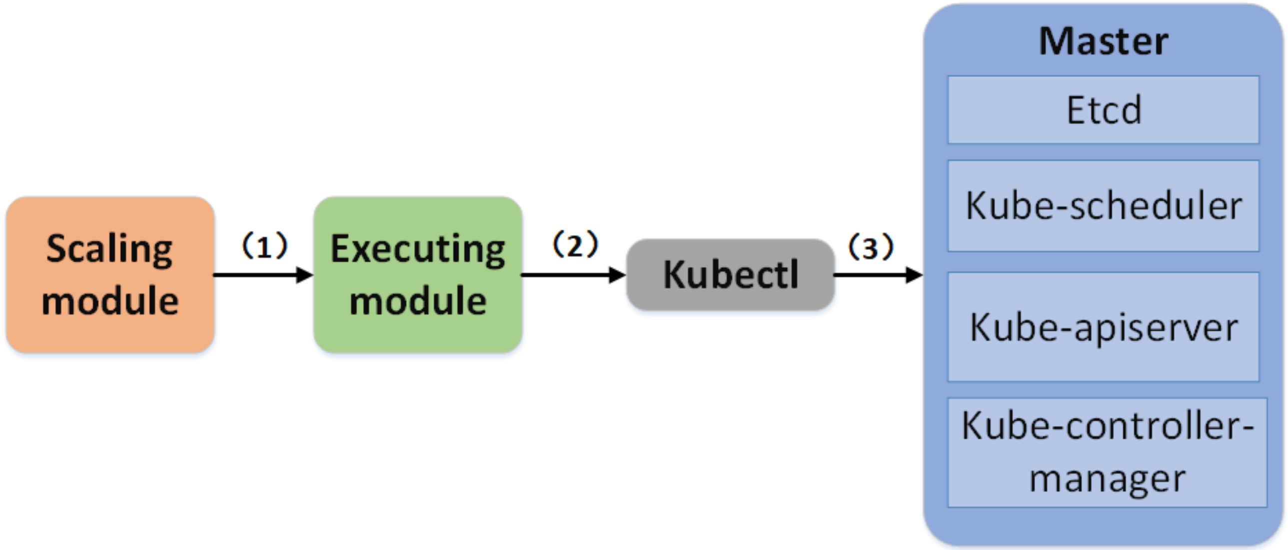
Scaling Module

- **Goal:**
  - Scale up or down according to the monitoring data from monitor module, while meet the QoS requirements by QoS module

- **Cluster Scaling Algorithm:**
  - If $U > U_{threshold}$,
    - $N_{add} = 2 * N_{add}$, if the cluster scaled up last time
    - $N_{add} = 1$, if the cluster don't scaled up last time
  - If $U < 40\%$,
    - $N_{remove} = 2 * N_{remove}$, if the cluster scaled down last time
    - $N_{remove} = 1$, if the cluster don't scaled down last time

Executing Module

- **Goal:**
  - **Implement each operation for cluster scaling based on the output of the scaling module**
    - **Generate specific command for Kubectl to realize the scaling operation**



Outline

- System design
- Evaluation
- Conclusion

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Experimental Setup

- **5 physical machines:**

  4-cores Intel(R) Core(TM) i5-4460S 2.9 GHz CPU, 4 GB memory and 1 TB disk

- **CentOS Linux release 7.5**
- **Kubernetes v1.10 and Docker v18.06-ce**

  Heapster v1.5.2 and InfuxDB v1.3.3

- **Testing application:**

  Ticket Monster, deployed in Deployment manner with the HorizontalPodAutoscaler

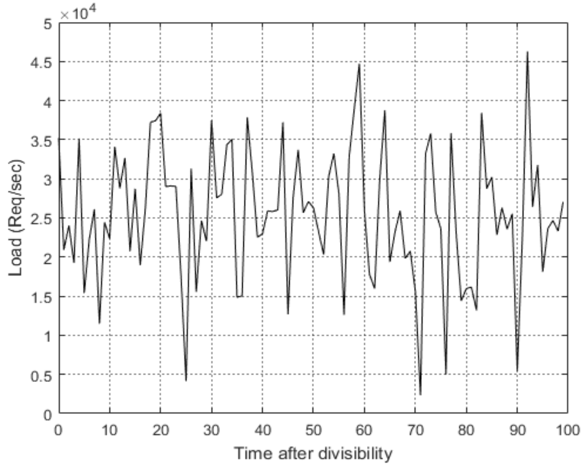
- **Workloads:**

  Apache JMETER , simulate the workload that users send HTTP requests to the Ticker Monster

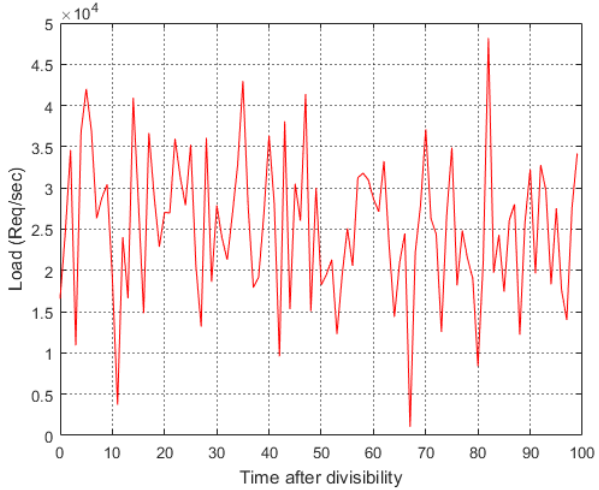
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Workloads

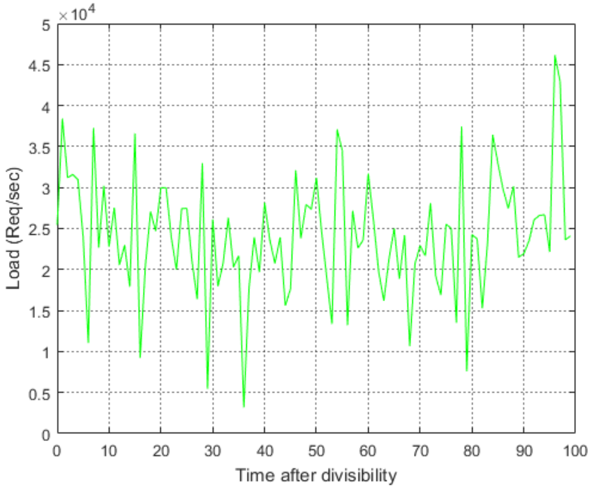
Workload Examples:



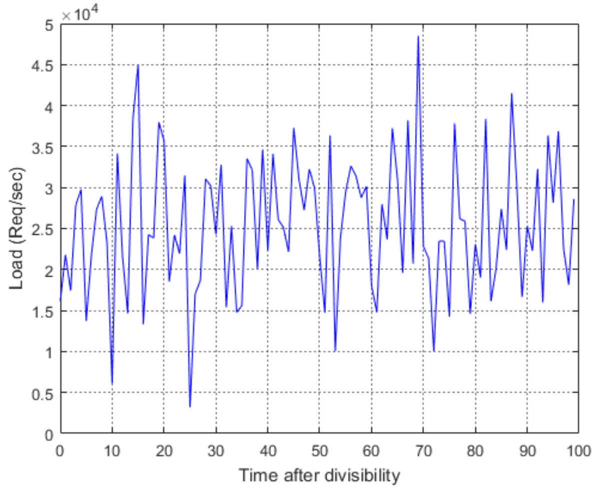
Load a



Load b



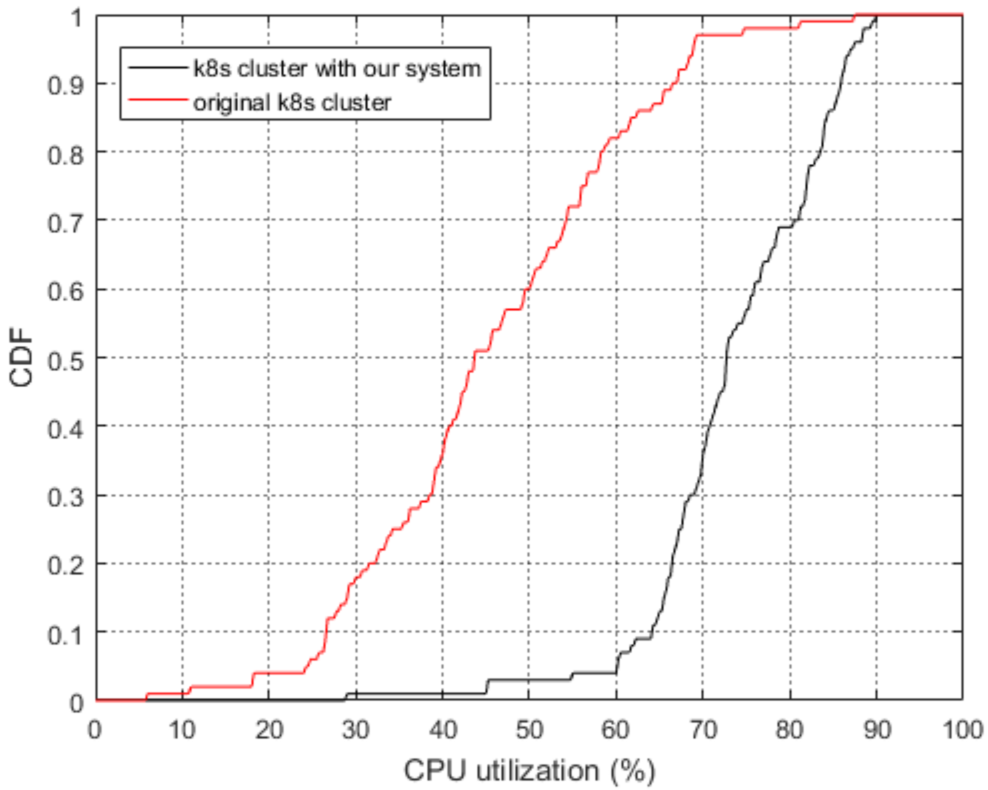
Load c



Load d

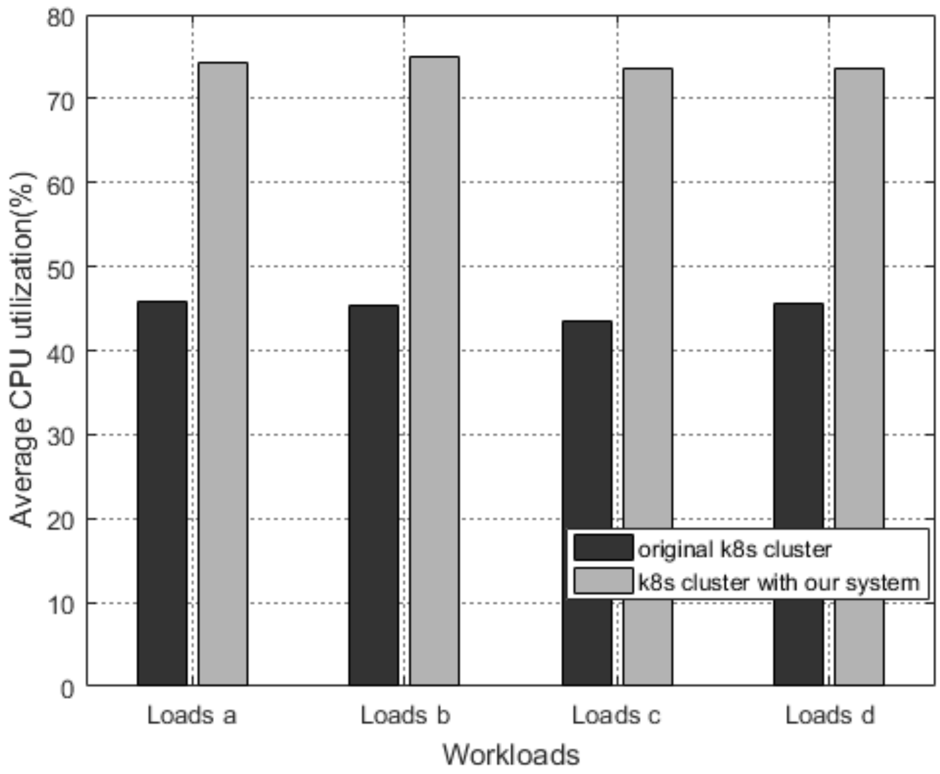
Improvement of CPU utilization

- **CDF of CPU utilization of the original Kubernetes cluster and the Kubernetes cluster with our system:**



Improvement of CPU utilization

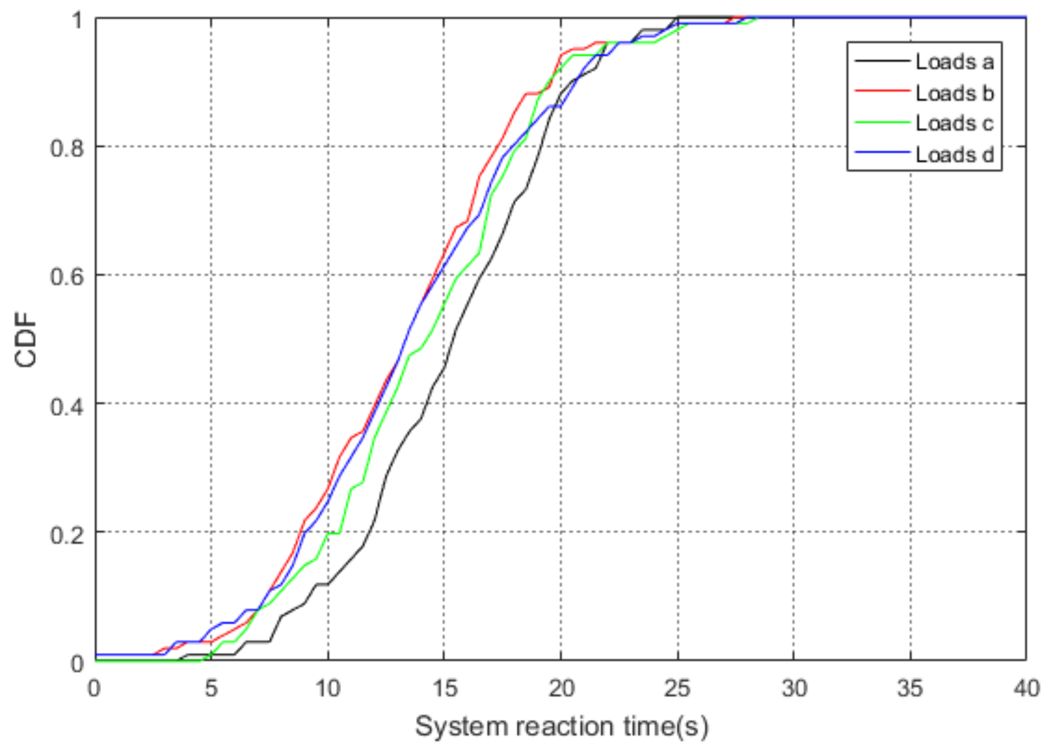
- The average CPU utilization of the original Kubernetes cluster and Kubernetes cluster with our system under four different workloads:



Improved by 28.99%

Reaction time

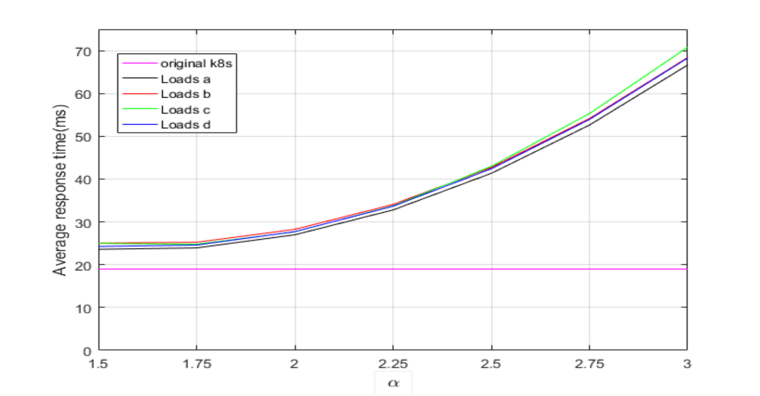
- CDF of the system reaction time under four different workloads:



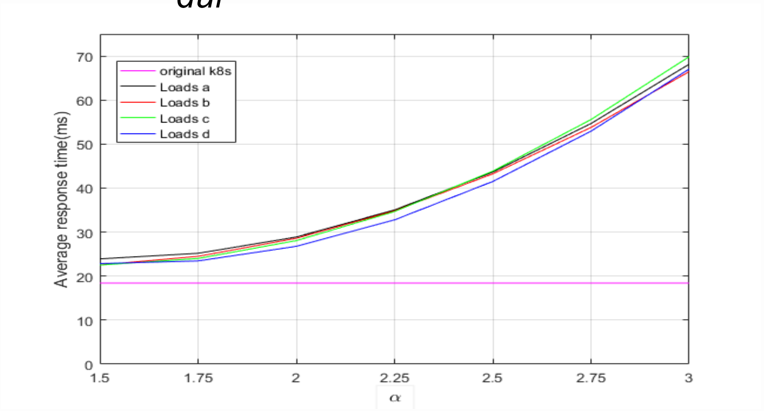
- The average reaction time of the system is about 15s.

Parameter Selection

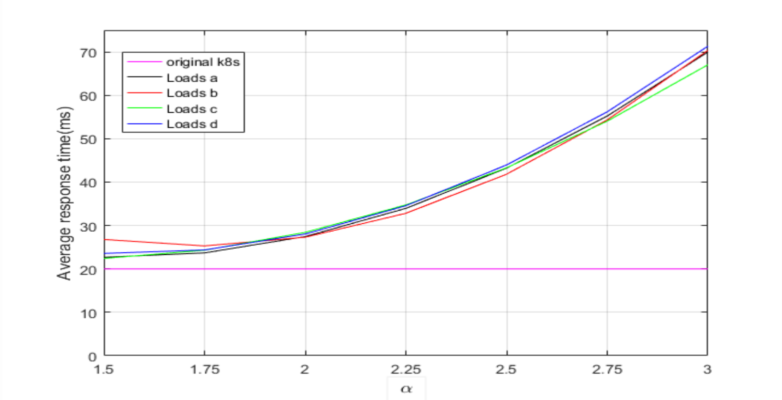
QoS coefficient α under different duration T_{dur}



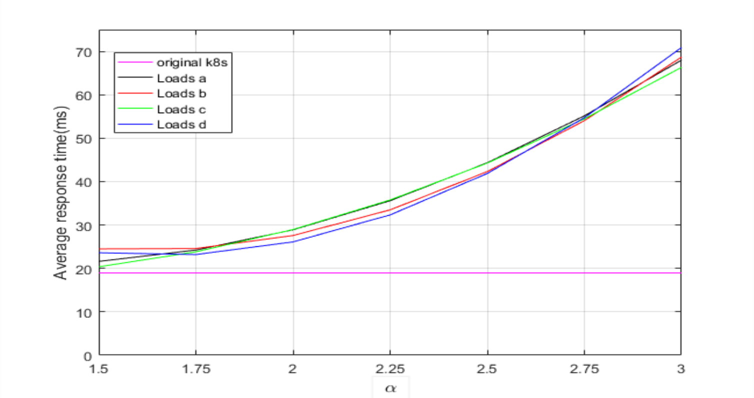
T_{dur}=10s



T_{dur}=20s



T_{dur}=30s



T_{dur}=40s

select α = 2

Outline

- System design
- Evaluation
- **Conclusion**

Conclusion

- **We propose a system, which dynamically adjusts scale of a Kubernetes cluster, to improve the resource utilization.**
- **The system can automatically derive a threshold of system resource utilization according to the specific application in a Kubernetes cluster, which promises QoS in a Kubernetes cluster.**
- **The experimental results show that CPU utilization of a Kubernetes cluster with our system is improved by 28.99% than that of a original Kubernetes cluster on average.**

Thank you！
Q & A