Right-Sizing and Auto-Scaling of MySQL Containers in Kubernetes

Yuan Chen

JD.com

京东

About JD.com

China's largest online and overall retailer and biggest Internet company by revenue

- 300 million+ active users
- 2018 revenue: $67.2 billion

China's largest e-commerce logistics infrastructure and unrivalled nationwide fulfillment network

- 550+ warehouses
- Covering 99% of population
- Standard same-and next day delivery

First Chinese internet company to make the Fortune Global 500

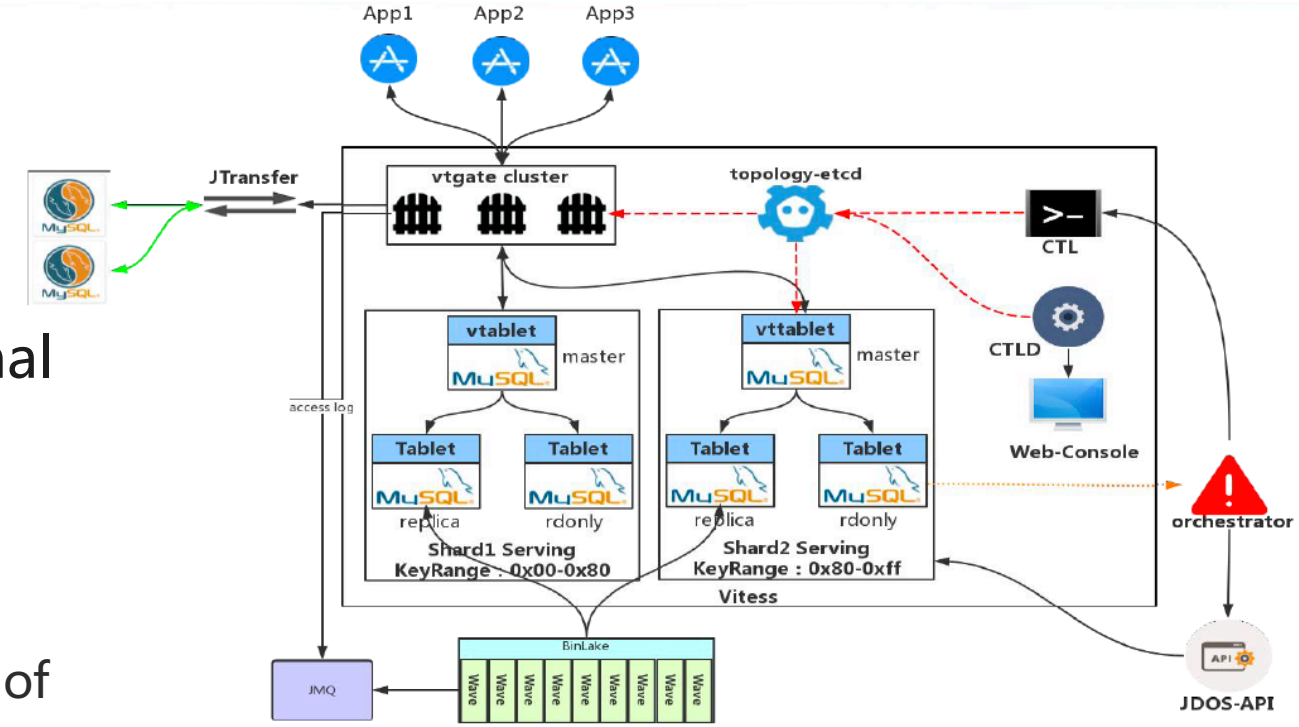
Strategic partnerships

Tencent Walmart Google

京东

JD Elastic Database

- MySQL for key businesses by serving large volumes of complex transactional data.
- MySQL databases in containers on JD Kubernetes platform
  - Thousands of physical serves and tens of thousands of containers
- Vitess for scalable management and scaling.



MySQL kubernetes Vitess

One of the largest Vitess deployments

Problems and Challenges

- Difficult to estimate the resource demand
- Dynamic demand
- **Performance guarantee**
- Multiple dimensional resources: CPU, Memory, **Disk, I/O Bandwidth**
- **Stateful applications with a lot of data**

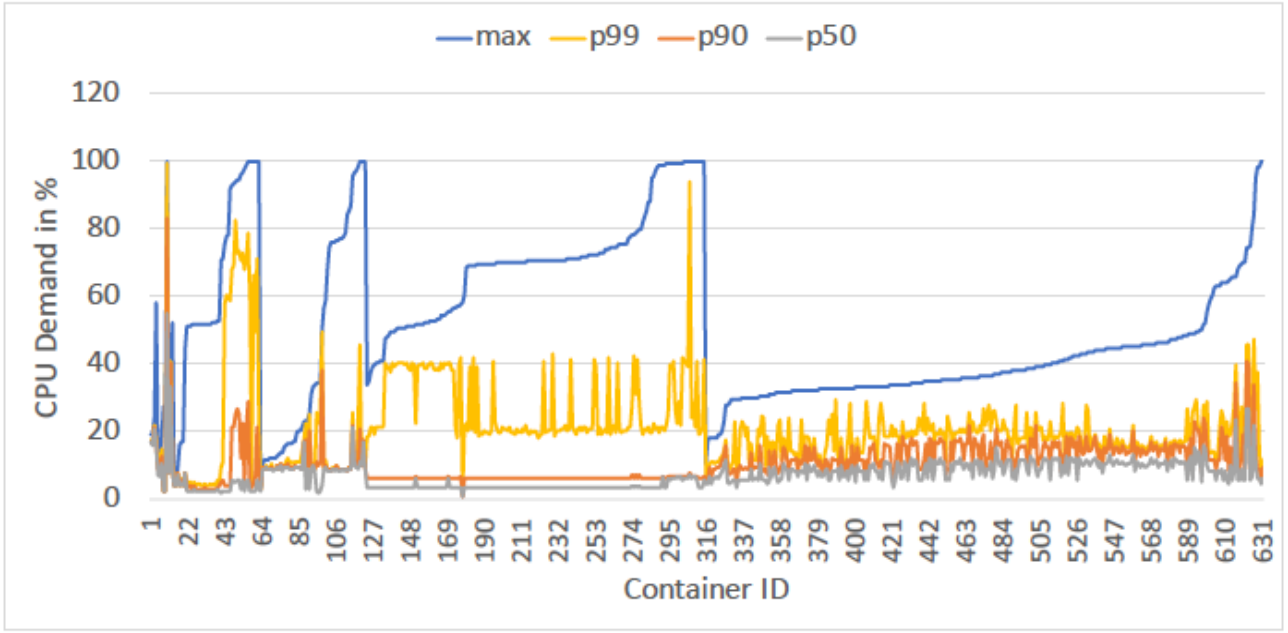
Workload Characterization

7 database systems and 631 containers

Application Name	Containers#	Days# (minutely)
DongDong	4	51
DistributedLog	6	51
OrderCancel	3	51
OrderTrace	49	42
TrainTicket	61	51
MallBill	192	51
SKUPrice	316	51

Table 1: Application details.

CPU Utilization



Key observations:

1. The 90–percentile utilization is about half or less of the maximum utilization.
2. Sizing based on 90-percentile utilization instead of the maximum utilization could lead to significant savings.

Right Sizing

- Estimate the workload demand

  New workload

  – Resource requirements

  – Meta-data based classification + group resource requirements

  Existing workload

  – Historical usage analysis + prediction (ARIMA, LSTM, …)

- Sizing based on percentile and correlation between workloads

  – Use shared headroom to cope with peak demands

  – Adjust both request and limit

Right Sizing of CPU Resources

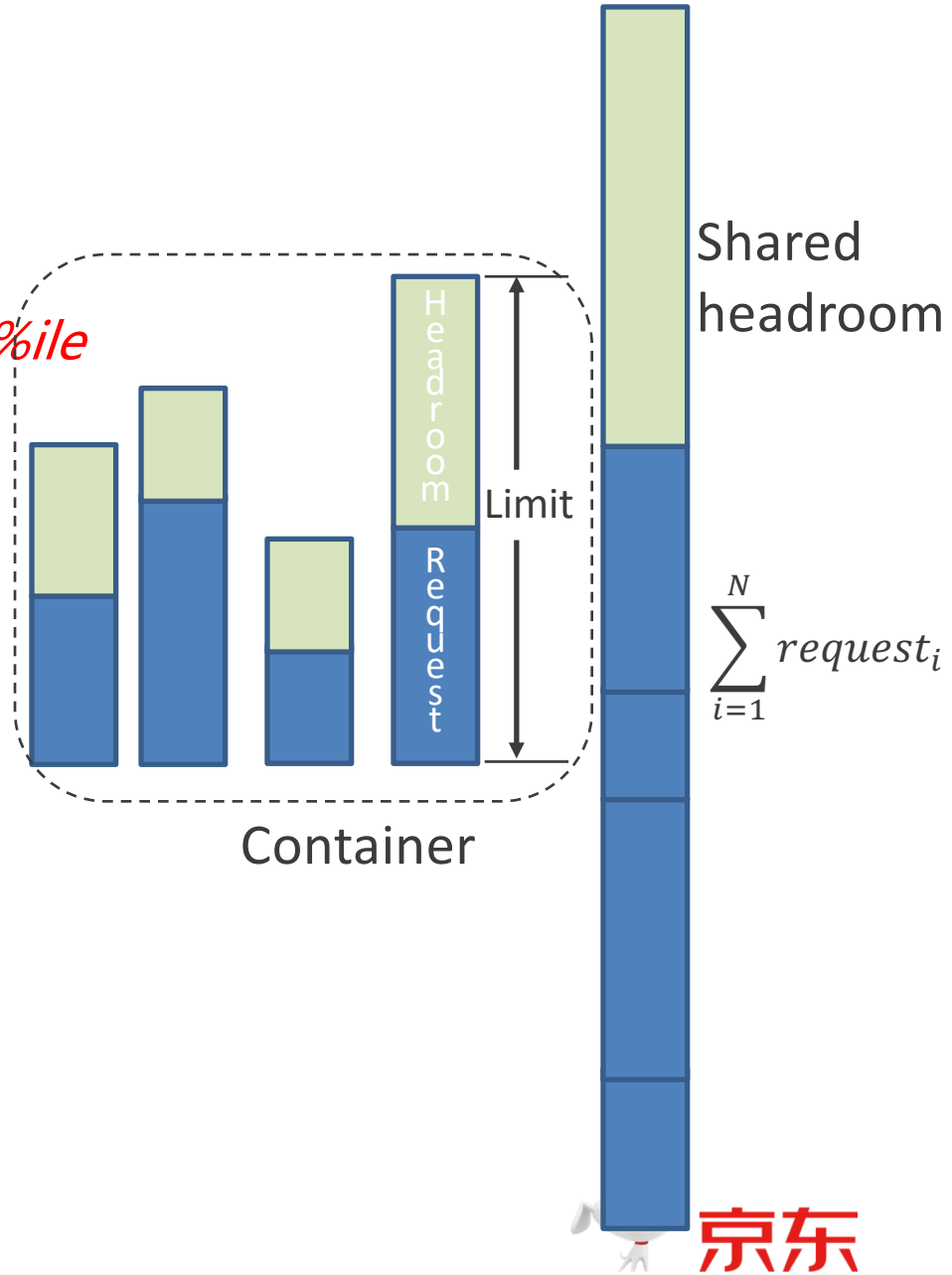
Container

- Tail bound based guaranteed resource: *request = x%ile*
- Maximum resources : *limit = y%ile*
- Headroom: *headroom = limit – request*

Host

- Request: *Request* = $\sum_{i=1}^{N} request_i$
- Shared headroom
  - *Headroom<=* $\sum_{i=1}^{N} headroom_i$
- Resource constraint:

  Request + Headroom <= Capacity



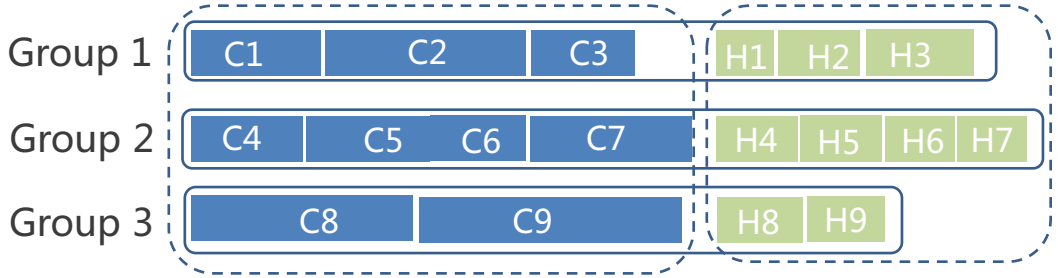
Right Sizing of CPU Resources

Request : $Request = \sum_{i=1}^{N} request_i$

Shared headroom

- Non-correlated containers:
  $Headroom = MAX_i\,(limit_i - request_i)$
- Correlated containers:
  $Headroom = SUM_i\,(limit_i - request_i)$

Constraints: *Request + Headroom <= Capacity*

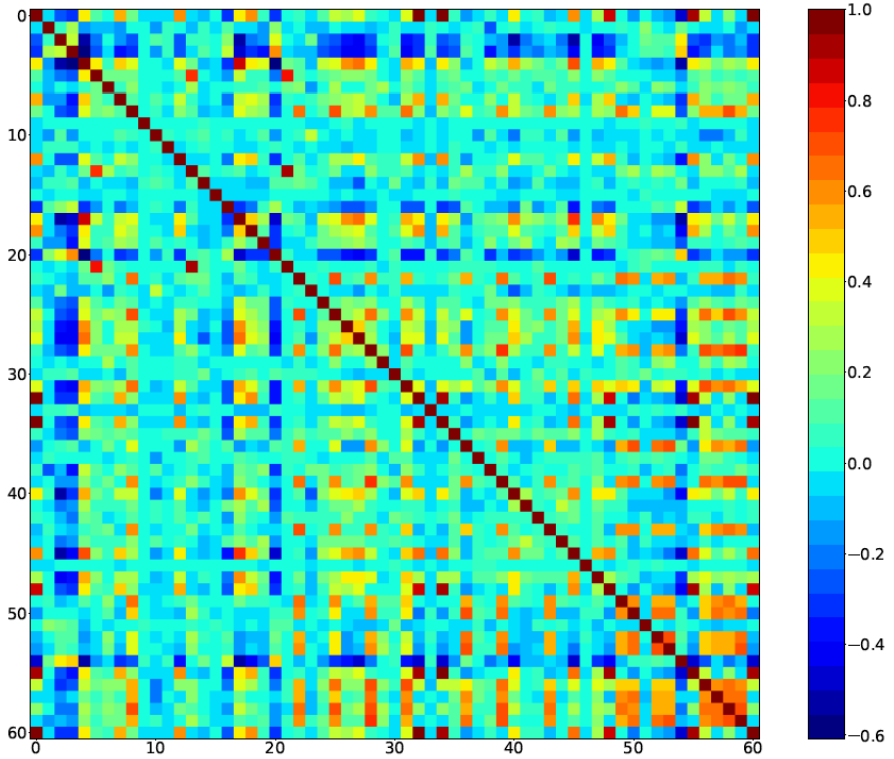


Request = C1+C2+…+C9 **Headroom** = MAX (H1+H2+H3, H4+H5+H6+H7, H8+H9)

Total = C1+C2+…+C9+H4+H5+H6+H7

max_of_sum < sum_of_max

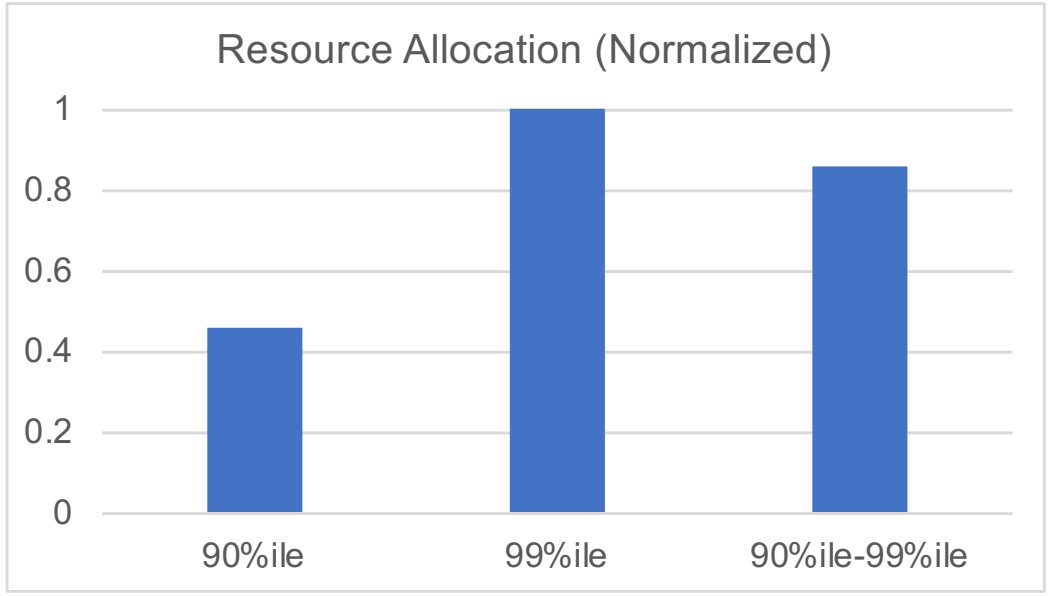
Correlation Analysis

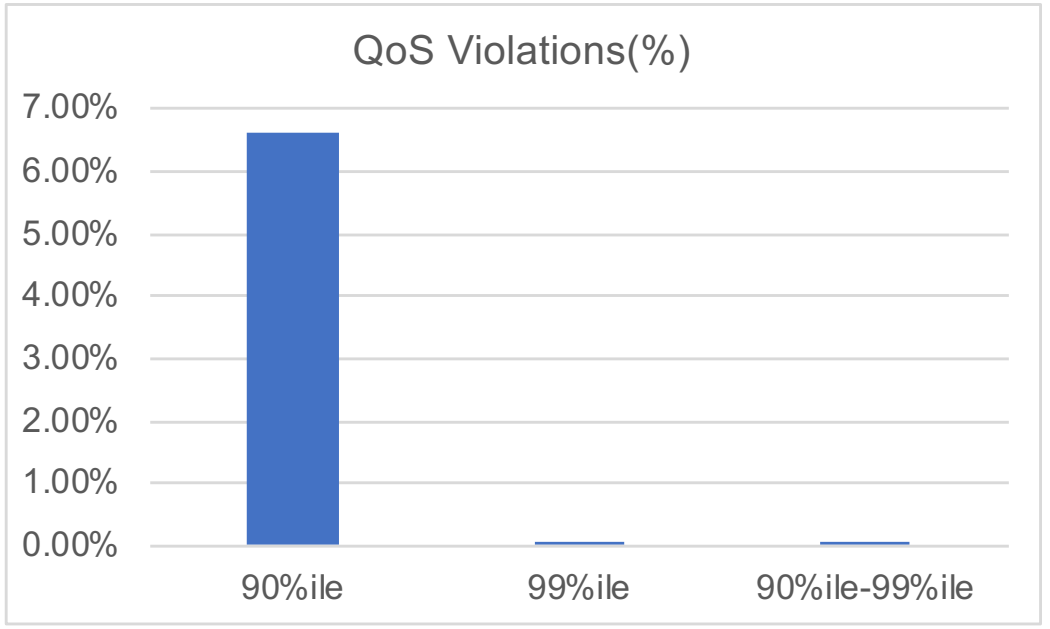


A large number of containers do not have strong correlation!

Experimental Evaluation

Comparison of Different Sizing Methods



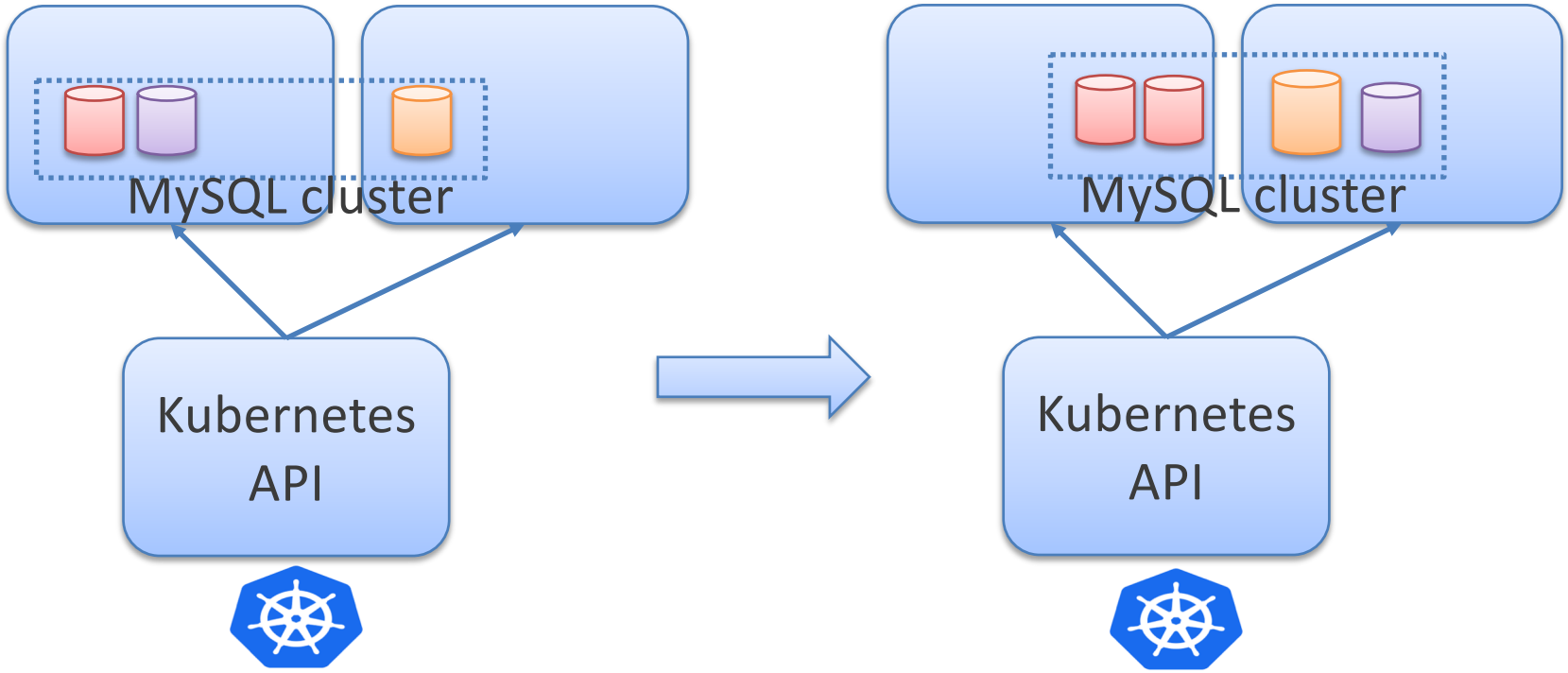


Randomly schedule 18 containers among 563 containers

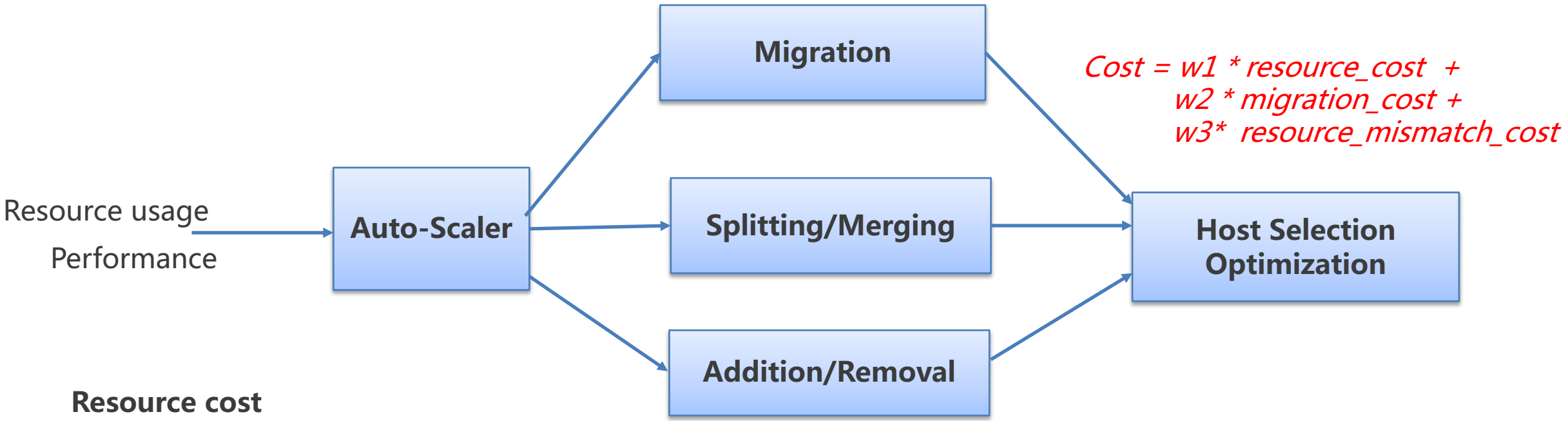
京东

Auto Scaling and Rescheduling

- Split or merge instances
- Add or remove instances
- Migrate or reschedule



Auto-Scaling: Overview and Cost Models



Resource cost

$$\sum_{i=1}^{M} C_i\,(90\%ile) + \max_i \sum_{j=1}^{N_i} [C_{i,j}\,(99\%ile) - C_{i,j}\,(90\%ile)]$$

Migration cost: *k * database size*

Resource usability: *balancing, availability, anti-affinity*

Host Selection: Multi-Resource Balance

K8S: BalancedResourceAllocation = 10 – abs(cpuRemaingFraction-memoryRemainingFraction)*10

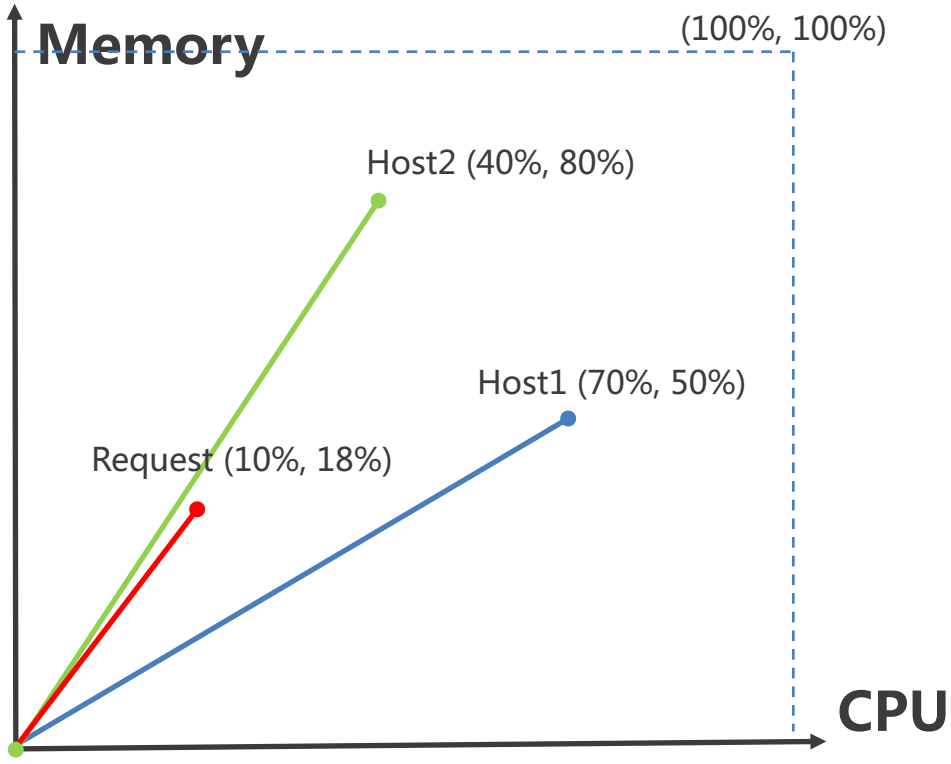
Optimization：similarity between the request resource and available resources

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Host available resources： (U_{CPU}, U_{MEM})

Pod resource request： (R_{CPU}, R_{MEM})

Metric: $$\frac{(R_{CPU}U_{CPU} + R_{MEM}U_{MEM})}{\sqrt{R_{CPU}^2 + R_{MEM}^2}\sqrt{U_{CPU}^2 + U_{MEM}^2}}$$



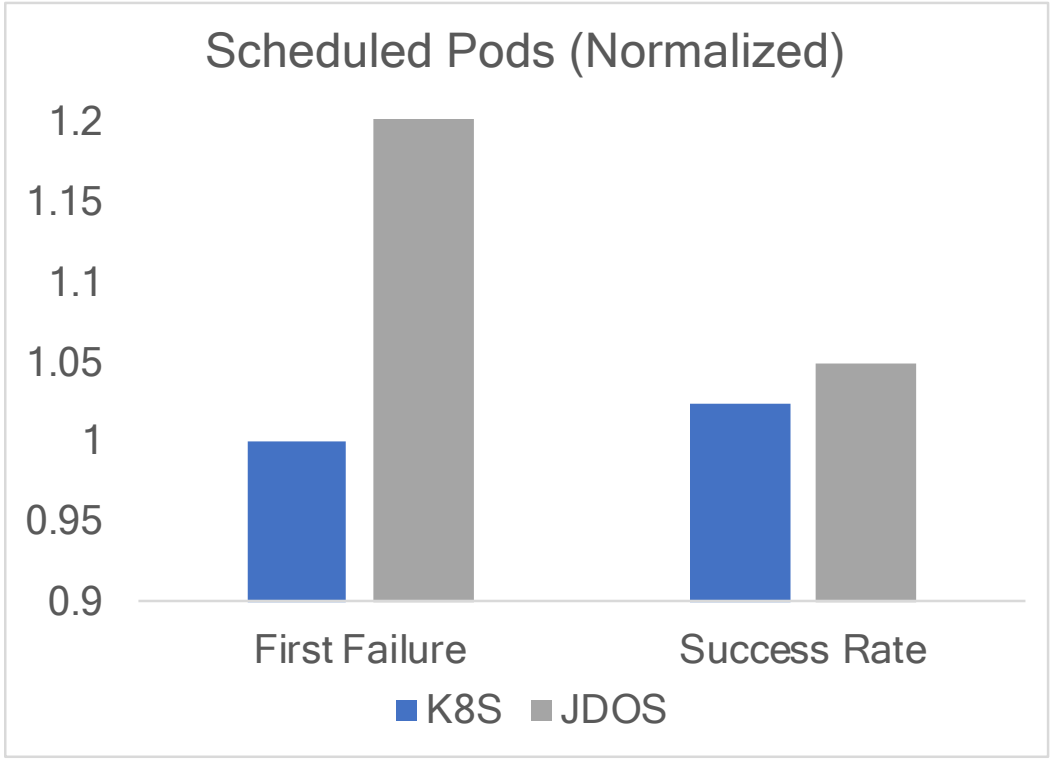
Experimental Evaluation

Setup

- 4,857 servers (8 configurations)
- 25,000 containers (120 configurations)

Results

- **Schedule 5%-20% more containers than K8S**
- **Comparable performance with K8S**



Host Selection: Resource Availability

K8S :

LeastRequestedPriority = cpu((capacity – sum(requested)) 10 / capacity) + memory((capacity – sum(requested)) 10 / capacity) / 2

Optimization: dynamic variable weights

$$w_{cpu} * \text{cpuAvailFraction} + w_{mem} * \text{memAvailFraction}$$

Algorithm 1:

Weighted sum of CPU : $\text{cpuFraction} = \sum_i f_i * \text{cpuFraction}_i$

Weight sum of memory : $\text{memFraction} = \sum_i f_i * \text{memFraction}_i$

$$w_{cpu} = \frac{\text{cpu}Fraction}{cpuFraction + memFraction}$$

$$w_{mem} = \frac{memFraction}{cpuFraction + memFraction}$$

Algorithm 2: Hosts with stable resource usage are more usable.

MAD- Median Absolute Deviation

$$w_{cpu} = 1 - \text{median}(|\text{cpuUtil}_i - median(\text{cpuUtil}_i)|)$$

$$w_{mem} = 1 - \text{median}(|\text{memUtil}_i - median(\text{memUtil}_i)|)$$

京东

Host Selection: Correlation-awareness

X : existing pods resource usage

Y : new pod resource usage

Anti-affinity: If Y can be predicted by Y, X and Y are highly correlated. We should avoid placing them on the same host.

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,k} & \cdots & x_{1,K} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,k} & \cdots & x_{n,K} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & \cdots & x_{N,k} & \cdots & x_{N,K} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \\ \vdots \\ y_N \end{bmatrix}$$

Multi-regression analysis $\mathbf{b} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} \quad \hat{\mathbf{y}} = \mathbf{X}\mathbf{b}$

Correlation $$\frac{\sum_{i=1}^{n}(\mathbf{y}_i - m_Y)^2(\hat{\mathbf{y}}_i - m_{\hat{Y}})^2}{\sum_{i=1}^{n}(\mathbf{y}_i - m_Y)^2 \sum_{i=1}^{n}(\hat{\mathbf{y}}_i - m_{\hat{Y}})^2}$$

Key Results

Automated sizing system saved resources and cost

- CPU cores: several tens of thousands of CPU cores
- Memory: several hundreds of TB
- Disk: tens of PB
- Operation and management cost

Auto-scaling and rescheduling

- 5-20% more containers
- Performance guarantee

Conclusions

- Running MySQL in containers on Kubernetes and using Vitess cluster management is the foundation of MySQL resource optimization and management at JD.
- Statistical analysis and prediction based resource sizing and scaling approaches are effective to improve the resource utilization of containerized MySQL.
- **Kubernetes + Vitess + Advanced Resource Optimization Algorithms** significantly improves the resource efficiency and reduces the costs of running MySQL at JD.

Acknowledgements

Min Li

Haifeng Liu

Jiangang Fan

Fengcai Liu

Guangya Bao

Huasong Shan

An Peng

Yan Liu

京东

Thank you for your time!

Contact:

Yuan Chen
Email: yuan.chen@jd.com
WeChat: yuan_gt

京东

FYI

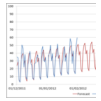
16:00-16:35 **Cost-Effective Scheduling of a Massive Number of Containers in Kubernetes**

ChubaoFS: a distributed file system for cloud native applications
https://github.com/chubaofs/chubaofs
SIGMOD 2019

System Architecture

JD.COM 京东

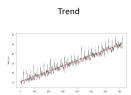
Time series of resource usage
(CPU, Memory, Disk, I/O)



ClickHouse

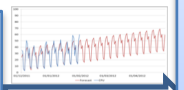
Trend Analysis

Demand without trends

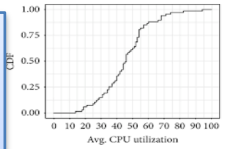


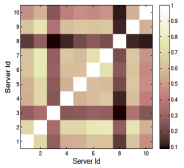
demand with trends

Forecast



Time Series Analysis





distribution, std, trend, periodicity, stability, correlation

Container

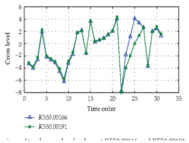
Tail-bound based Container Resizing

Server

Shared Headroom, Resource Balance, Correlation Analysis

Cluster

Rescheduling and Migration for Load Balancing and Consolidation



Real time Monitoring

Database Container Clusters

JDOS API

京东