# Tetris: Predictive Pod Placement Strategy for Kubernetes

Vasudev Bongale (vabongal)
Sohail Shaikh (sashaikh)
Kyle Martin (kdmarti2)

26-April-2019

# Agenda

- Problem Statement
- Idea
- Architecture
- Forecasting Strategy
    - Results
- Tetris-scheduler
    - Results
- Threat To Validity
- Limitations
- Future Work

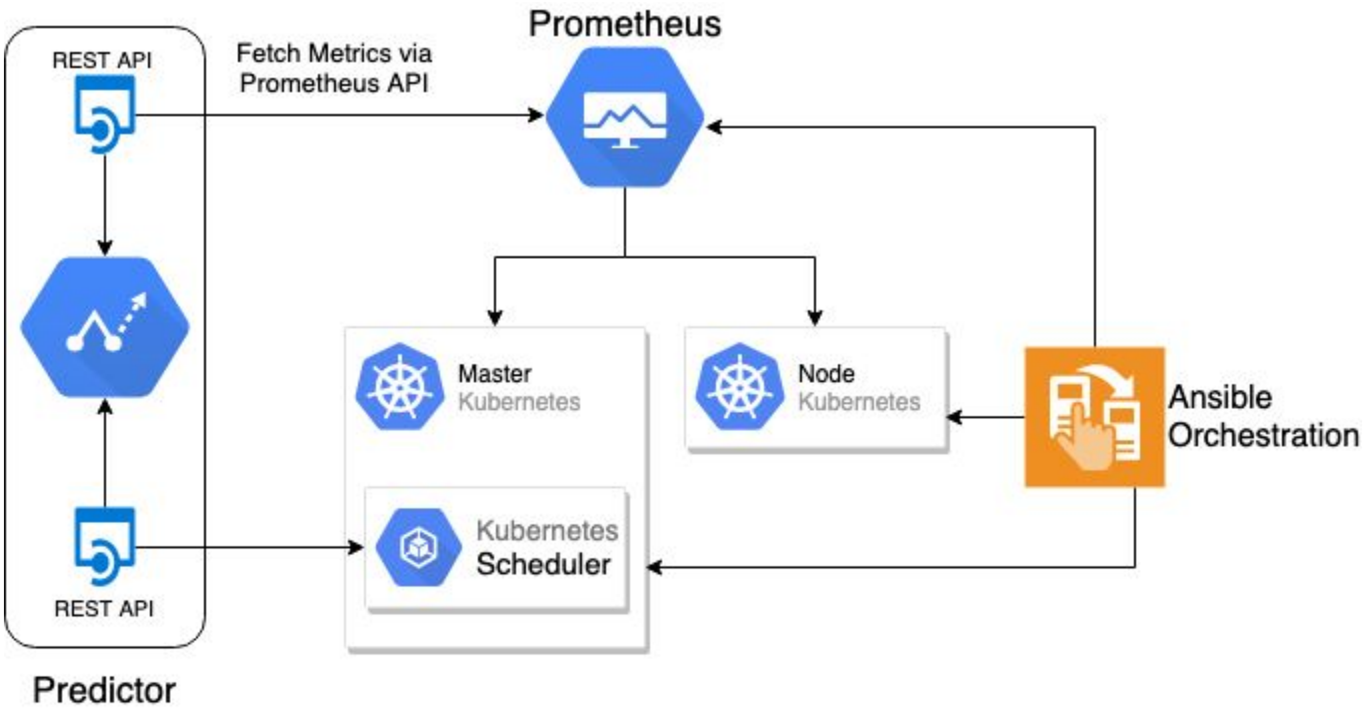
# Problem Statement

Kubernetes does not consider I/O utilization when provisioning pods. This could lead to performance degradation on an I/O bottleneck K8s Node. We propose to introduce a predictive pod placement strategy to avoid resource bottlenecks.

# Idea

- Schedule based on resource requirements.
  - Manually labelling pod by workload type.
  - Memory/IO intensive.
- Ensemble of prediction algorithms.
  - Intelligent placement.
  - Choose a node for a pod.

# Architecture



# Training Data Collection

1. Three different K8s stress-ng deployments.

| | |
|---|---|
| cpu | stress-ng --cpu $workers --cpu-load $cpu_load --cpu-load-slice $load_slice -t $time |
| mem | /usr/bin/stress-ng --vm $num_workers --vm-bytes 40% --vm-method rand-set --t $time_out |
| io | stress-ng --iomix $iomix --iomix-bytes 1g -t $timeout |

2. Stats exported by Prometheus Node Exporter.

# Prometheus Queries

- CPU Usage (%): "100 - (avg by (instance) (irate(node_cpu{job='k8s-nodes',mode='idle'}[2m])) * 100)"

- Used Memory (%): "100 * (1 - ((node_memory_MemFree{job='k8s-nodes'} + node_memory_Cached{job='k8s-nodes'} + node_memory_Buffers{job='k8s-nodes'}) / node_memory_MemTotal{job='k8s-nodes'}))"

- Disk I/O utilization (%): "100 * (rate(node_disk_io_time_ms{device="xvda", job='k8s-nodes'}[5m])/1000 or irate(node_disk_io_time_ms{device="xvda", job='k8s-nodes'}[5m])/1000)"

# APIs

- Components communicate using REST APIs.
- Prometheus Module
  - Get metrics from prometheus-server
  - Prometheus API
    - GET /api/v1/query
    - GET /api/v1/query_range
- Scheduler Queries a Flask API to interact with the module.

# Prediction Module

- Online Training
- Ensemble of prediction algorithms:
  – ARIMA
  – Holt Winters
  – WMA
- Training period is last 6 hours.

# Precompute Module

- Runs every 5 minutes to check the accuracies of models for cpu, io and memory utilizations.
- Picks the model with the least mean error percent and that model is used for prediction within that window.

# Auto Regressive and Integrated Moving Average

- Good for detecting seasonality.
- Expects a stationary series.
- Key concepts are order, lags and differencing:

Order: AR(p)

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_{Y\,t-2} + \ldots + \beta_p Y_{t-p} + \epsilon_t$$

ARMA(p, q) : q-> lag versions of forecast errors

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_{Y\,t-2} + \ldots + \beta_p Y_{t-p}$$
$$+ \epsilon_t + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-1}$$

ARIMA(p, d, q):

d = 0: no differencing

d = 1: performing differencing once

Apply differencing to remove trends from the series.

# Tuning ARIMA parameters:

- Choosing parameters (p, d, q) is difficult.
- Visual inspection is required to figure out seasonality and trend.
- We use AIC values as a criterion for choosing best. model using a grid searching technique.

# Holt Winter's Exponential Smoothing

- Can detect trends as well as seasonality.
- Popular because it's fast and cheap to compute.
- Key concepts: smoothing constants and updating equations.

Forecast = estimated level + trend + seasonality at most recent time point

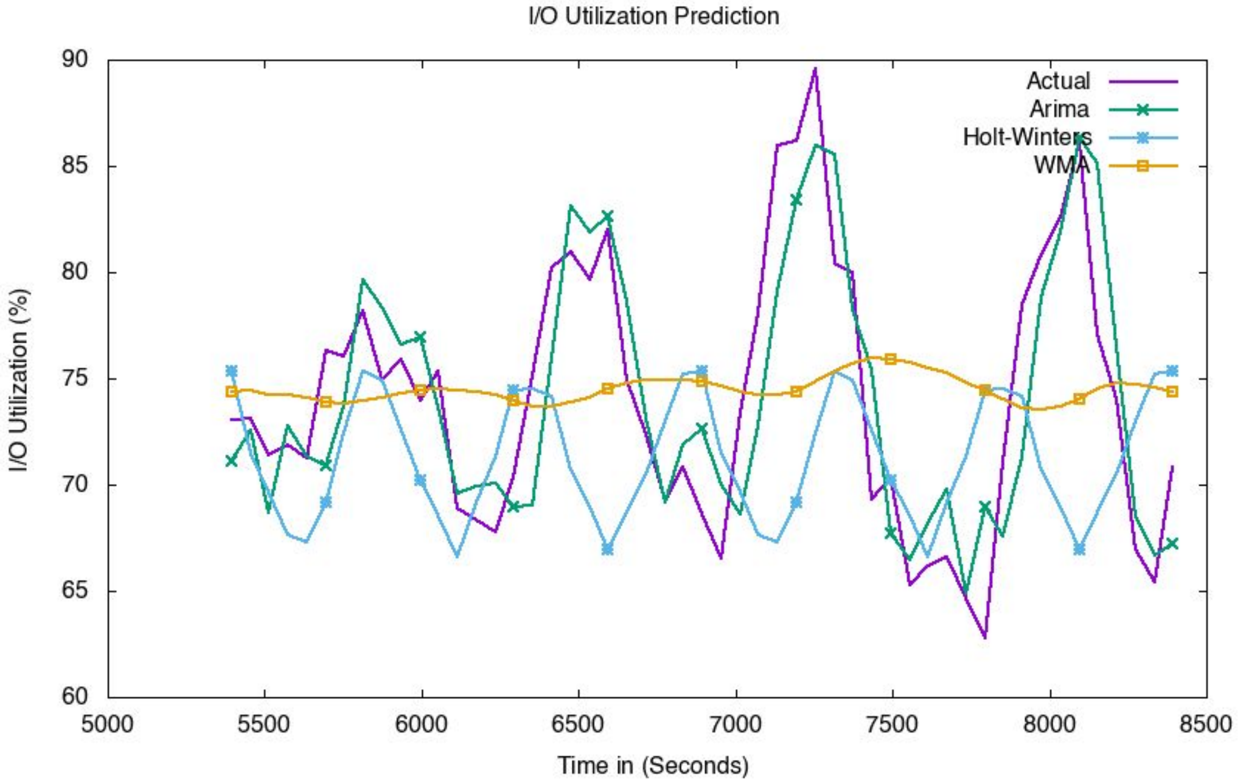
$$F_{t+k} = L_t + kT_t + S_{t+k-M}$$

$$Level(L_t) \quad trend(T_t) \quad Seasonality(S_t) \;\; with \;\; M \;\; seasons$$

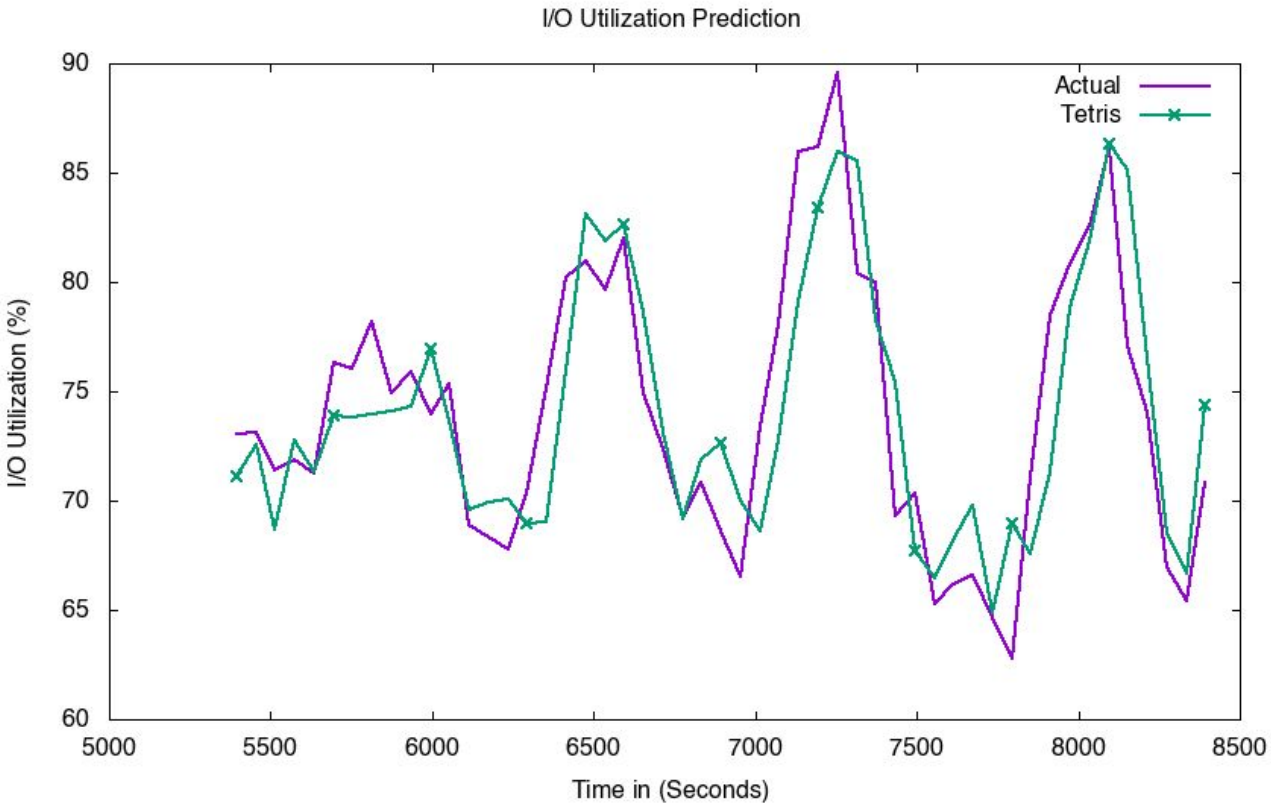
# Prediction Algorithm Analysis Methodology

1. Workload: I/O, Memory and CPU stressors (stress-ng).
2. Sampling rate: 10s
3. Training period: 6 hrs
4. Forecasting window: 6 points in the future (1 minute).
5. Shift prediction window by a minute.
6. Repeat 4, 5 for 1 hour.
7. This analysis was carried out in the offline mode.

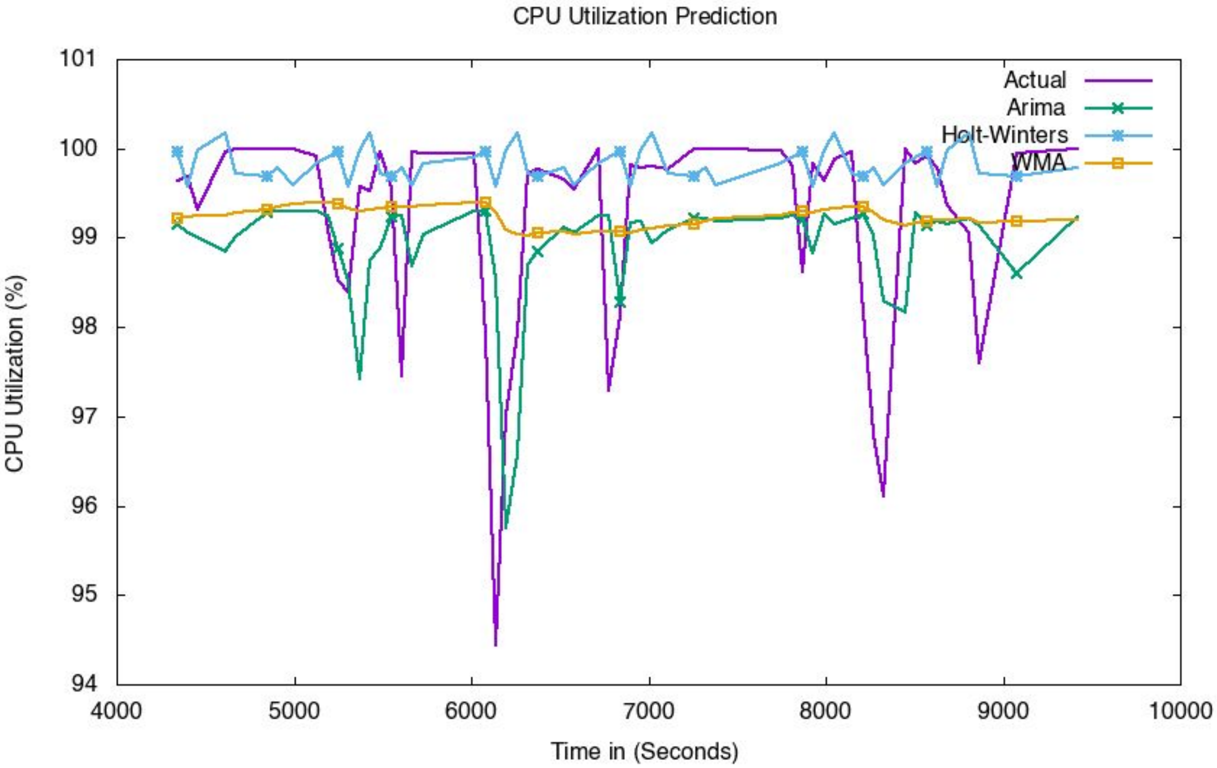
# Disk IO% Utilization



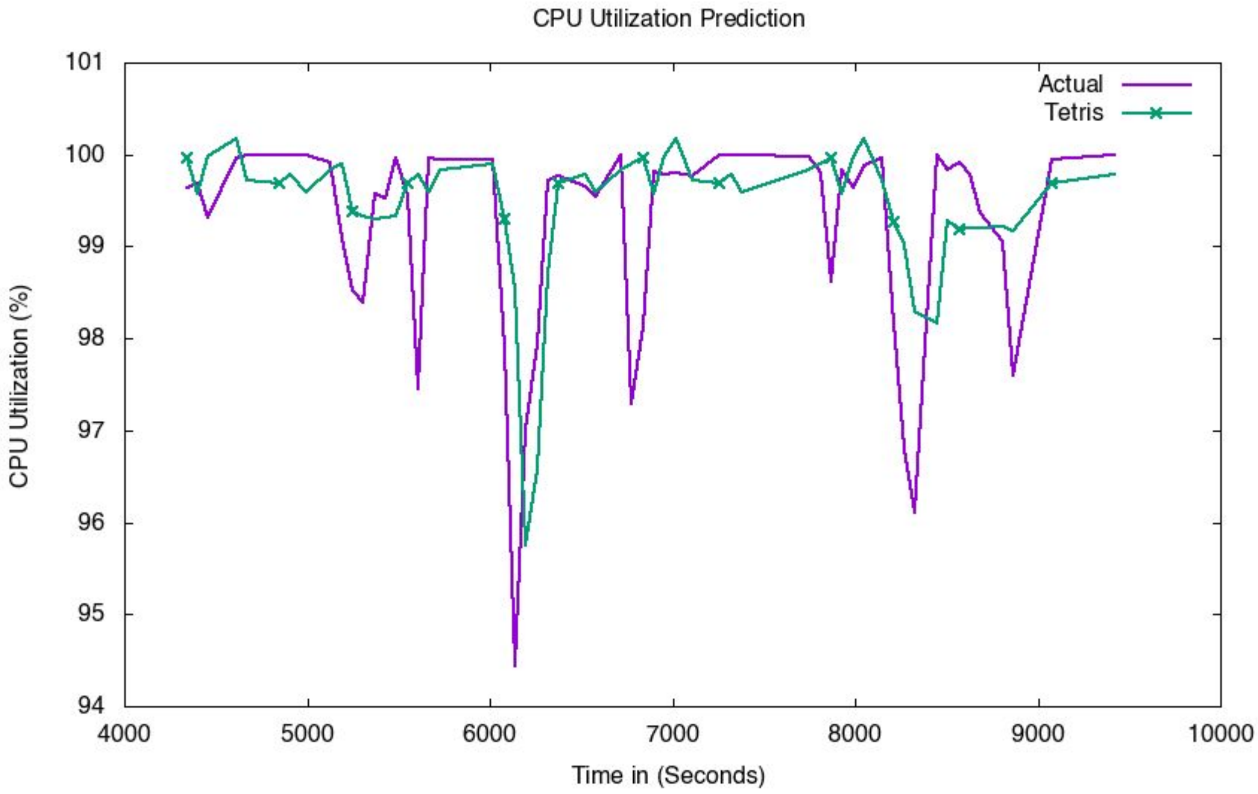
# Disk I/O Utilization Prediction using Tetris



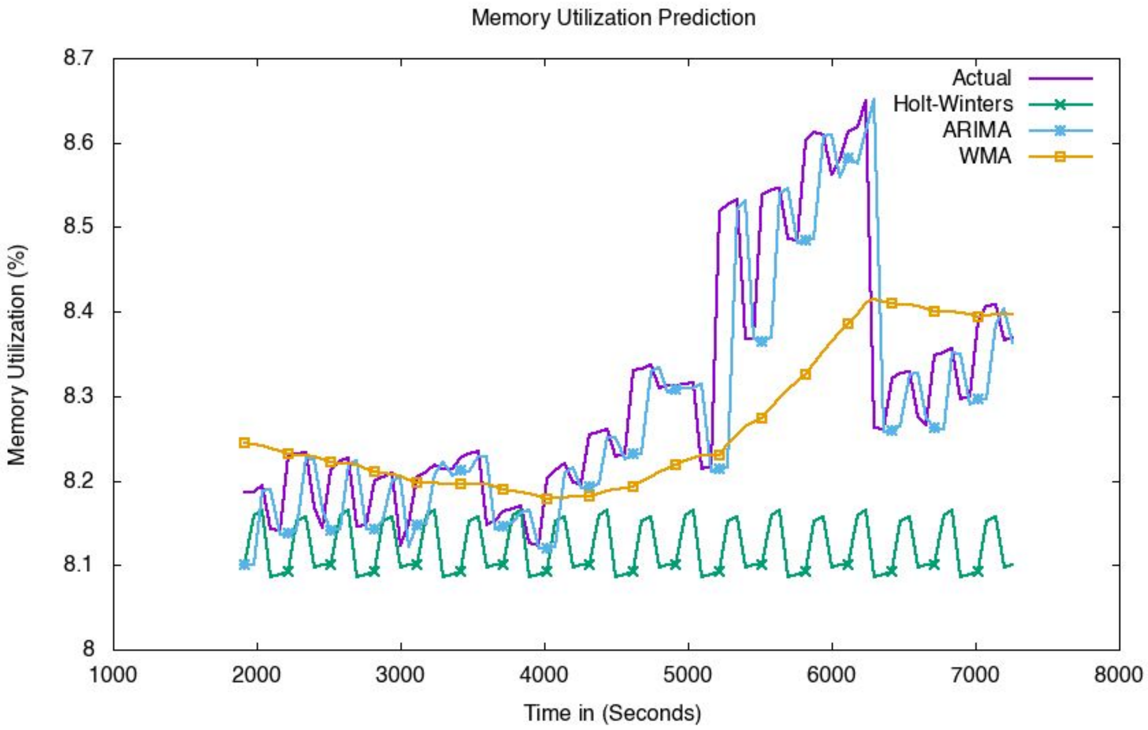
# CPU Usage % Prediction

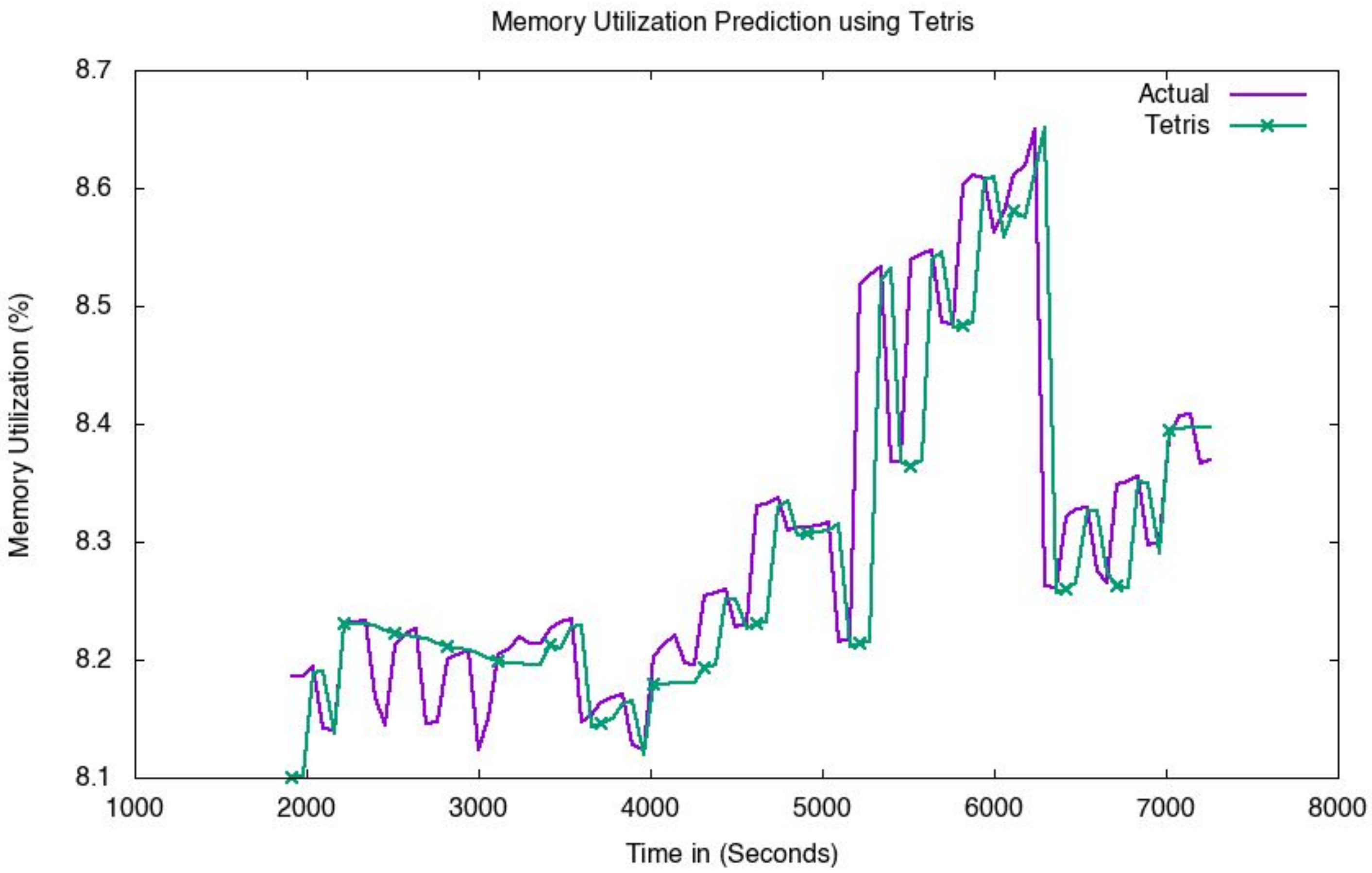


# CPU Utilization Prediction using Tetris



# Memory Usage %



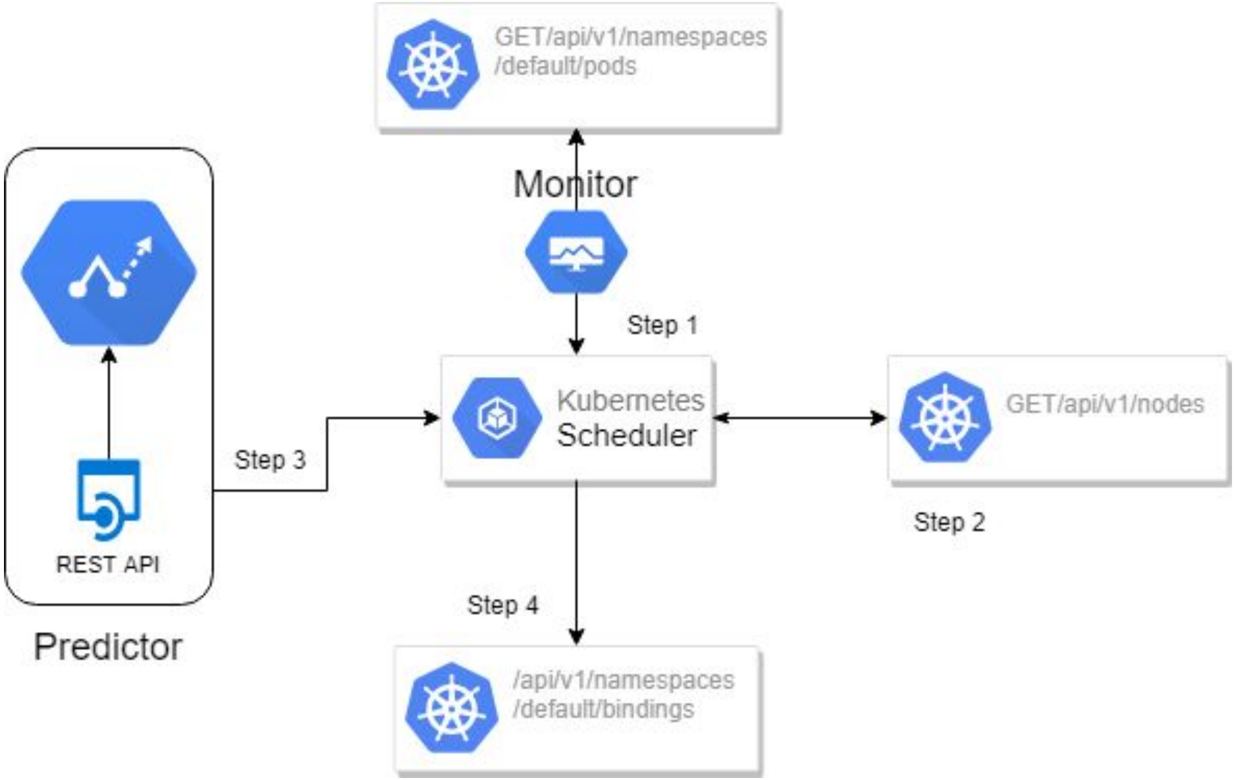


Memory Utilization Prediction using Tetris
Memory Utilization (%)
8.7
8.6
8.5
8.4
8.3
8.2
8.1
1000
2000
3000
4000
5000
6000
7000
8000
Time in (Seconds)
Actual
Tetris

# Prediction Mean Error Percent

| Algorithm | Disk I/O Utilization Accuracy | Memory Utilization Accuracy | CPU Utilization Accuracy |
|---|---|---|---|
| ARIMA | **3.5%** | 0.64% | 1% |
| Holt Winters | 7.2% | 2.1% | 0.8% |
| WMA | 6.6% | 1.05% | 0.9% |
| *Tetris | 3.65% | **0.55%** | **0.7%** |

# Tetris Scheduler



# Selecting A Scheduler

- Tetris-scheduler-io
- Tetris-scheduler-mem
- Tetris-scheduler-cpu

admin/sched/pod2.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: annotation-default-scheduler
  labels:
    name: multischeduler-example
spec:
  schedulerName: default-scheduler
  containers:
  - name: pod-with-default-annotation-container
    image: k8s.gcr.io/pause:2.0
```

```yaml
spec:
  schedulerName: tetris-scheduler-io
  containers:
    - image: mysql:5.6
      name: mysql
      env:
```

https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/#specify-schedulers-for-pods

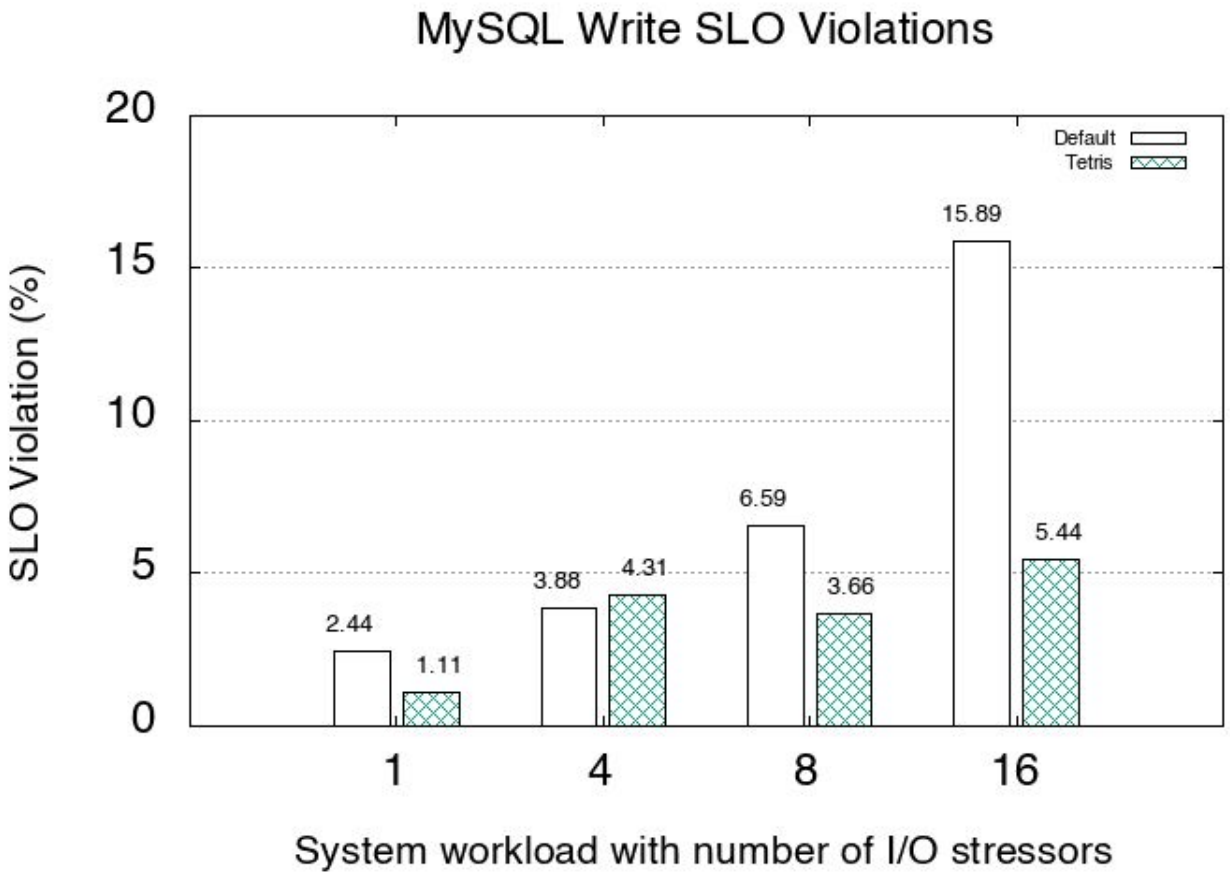
# SLO MySql Response Time Methodology

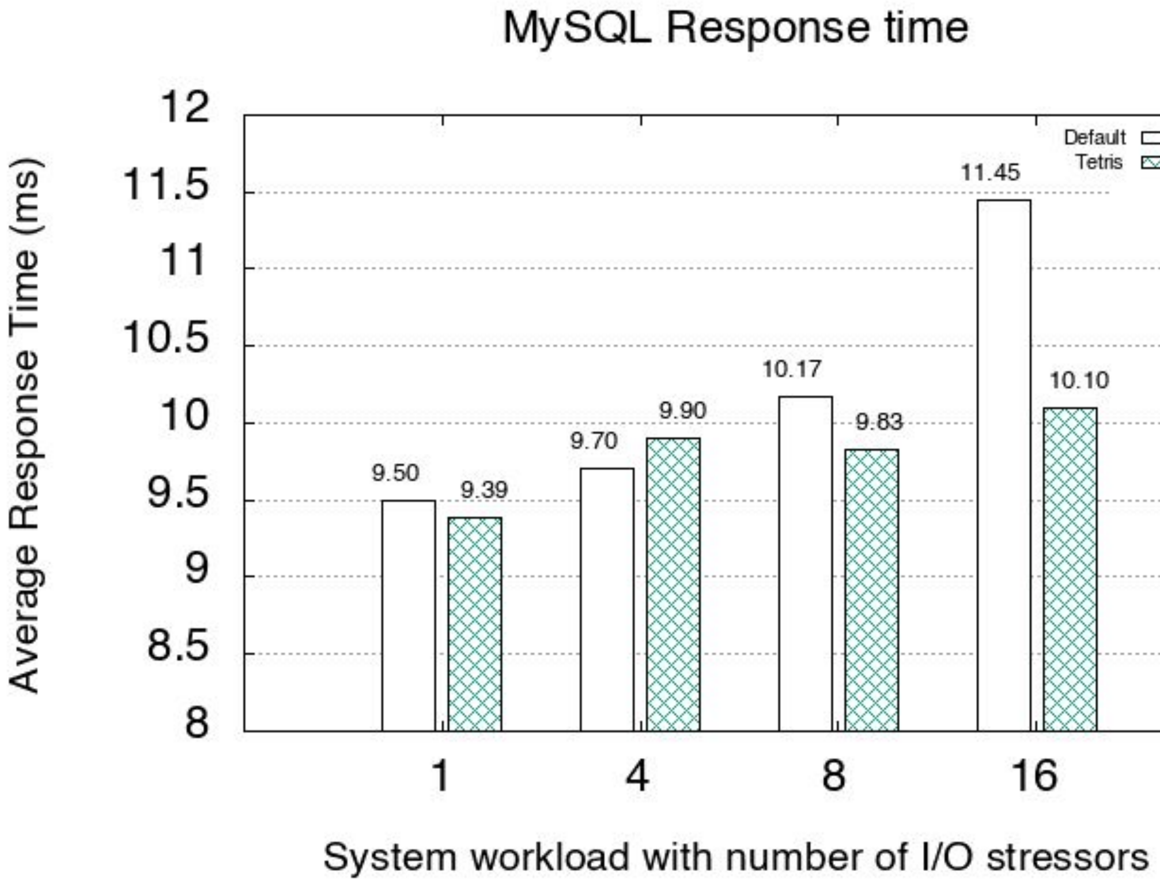
1. Deploy K8s MySQL pod.
2. Record response time of 10K insert requests (write).
3. Record response time of 10K query requests (read).
4. Determine 99th percentile respectively.

| | |
|---|---|
| **Write** | 11.84 ms |
| **Read** | 28.47 ms |

# IO Aware System Evaluation Methodology

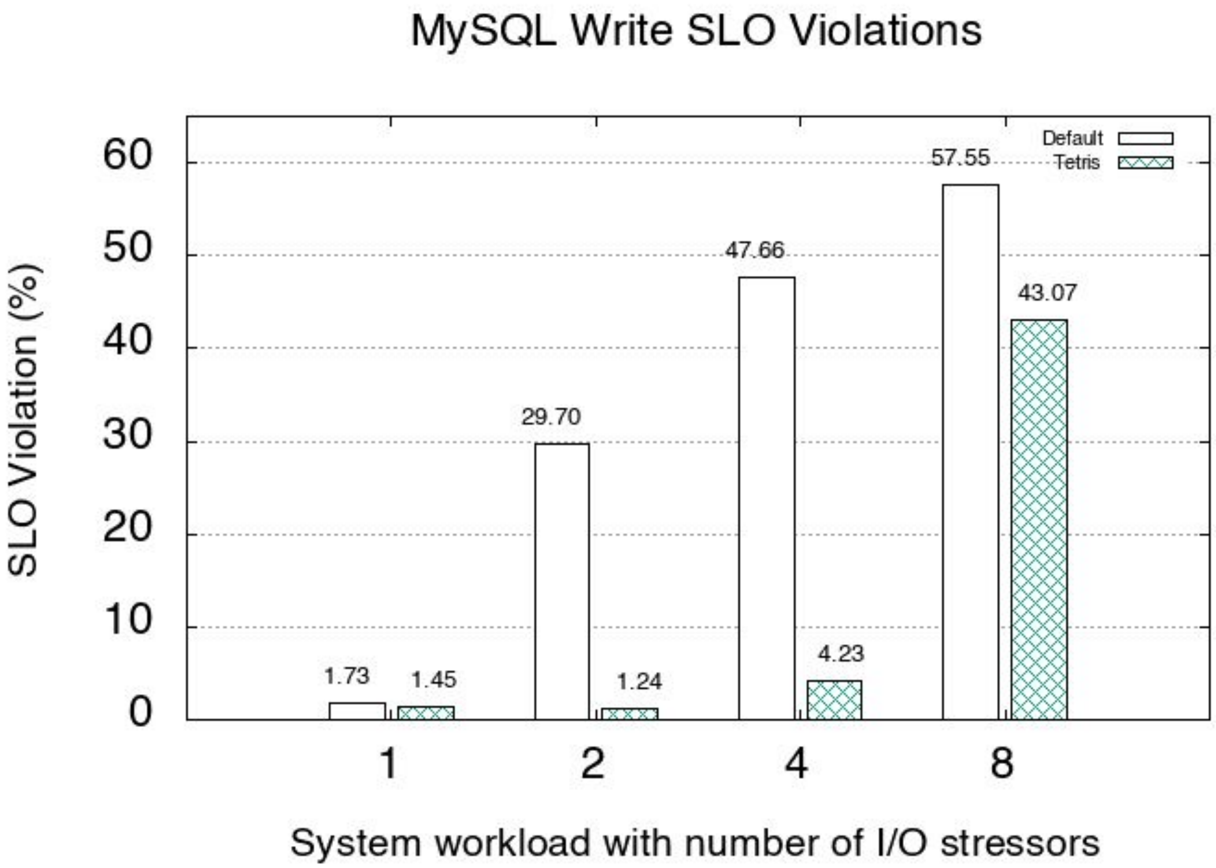
1. Deploy io-stressors.
2. Deploy MySql K8s deployment.
3. Record response time of 10K insert requests.
4. Record response time of 10K query requests.
5. Repeat 1-4 for K8s scheduler and Tetris.

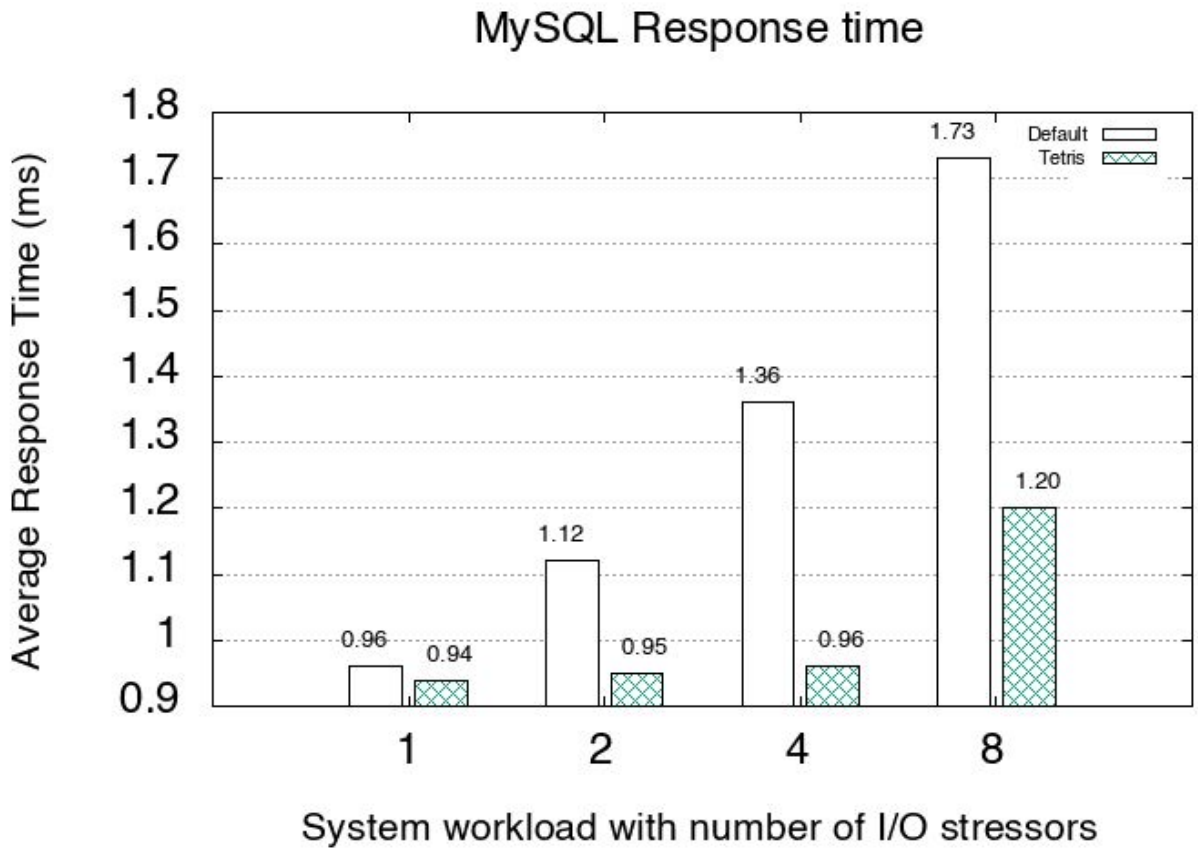

MySQL Write SLO Violations
SLO Violation (%)
20
15
10
5
0
Default
Tetris
2.44
1.11
3.88
4.31
6.59
3.66
15.89
5.44
1
4
8
16
System workload with number of I/O stressors


MySQL Response time
Average Response Time (ms)
12
11.5
11
10.5
10
9.5
9
8.5
8
Default
Tetris
9.50
9.39
9.70
9.90
10.17
9.83
11.45
10.10
1
4
8
16
System workload with number of I/O stressors

# Memory Aware System Evaluation Methodology

1. Deploy mem-stressors.
2. Deploy MySql K8s deployment.
3. Record response time of 10K insert requests.
4. Record response time of 10K query requests.
5. Repeat 1-4 for K8s scheduler and Tetris.


MySQL Write SLO Violations
SLO Violation (%)
60
50
40
30
20
10
0
Default
Tetris
57.55
47.66
43.07
29.70
4.23
1.73
1.45
1.24
1
2
4
8
System workload with number of I/O stressors


MySQL Response time
Average Response Time (ms)
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0.9
Default
Tetris
0.96
0.94
1.12
0.95
1.36
0.96
1.73
1.20
1
2
4
8
System workload with number of I/O stressors

# Threat to Validity

- CPU credits in AWS - allows burstable performance*

- Data Influence of past experiments
  - Data is not representative of actual workload.

* https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances.html

# Limitations

- Does not react well with fast pod placement requests.
- If the workload does not follow a pattern prediction node placement may not be accurate.

# Future work

- Consolidation of containers based on different workloads.
    - Leveraging PAC paper for pod signature extraction.

- Background Grid searching to optimize algorithms.

# Questions?