Predictive Auto-scaling in the Kubernetes Cluster Manager

THANK YOU

- Professor Albrecht, Professor, Williams College
- Brendan Burns, Lead Engineer for Kubernetes, Google

1.
Goals

Why do we care?

General Goal

Contribute to distributed system’s ability to reliably and resourcefully do large, varying amounts of computation.

A typical use case

Seek to have hbogo.com reliably and resourcefully served during the season premier of Silicon Valley...

But who will serve it and how?

Accomplishing General Goals

How do cluster managers reliably and resourcefully perform large, varying amounts of work?

What is a cluster manager?

- A cluster is a collection of commodity computers linked by a local-area network.
- A cluster manager is like an operating system for a cluster.

Benefits of Cluster Managers

Cluster managers allow us to perform computational work that could never be performed on a single computer.

Kubernetes

Specific Goal

To maximize the sum of two Kubernete’s metrics: Efficient Resource Utilization and Quality of Service

Unpacking this Goal

Kubernetes

An open-source cluster manager from Google.

Efficient Resource Utilization (ERU)

Is the application efficiently using the resources it is given?

Quality of Service (QOS)

Is the application accomplishing its stated purpose?

The goal is balance.

Increasing ERU/QOS while decreasing the other is easy, we seek to increase the summation.

Kubernetes specific terms

Pods

A stateless, replicable wrapper around related containerized applications (ex. a pod for hbogo.com contains an Apache web server and a cache of video content)

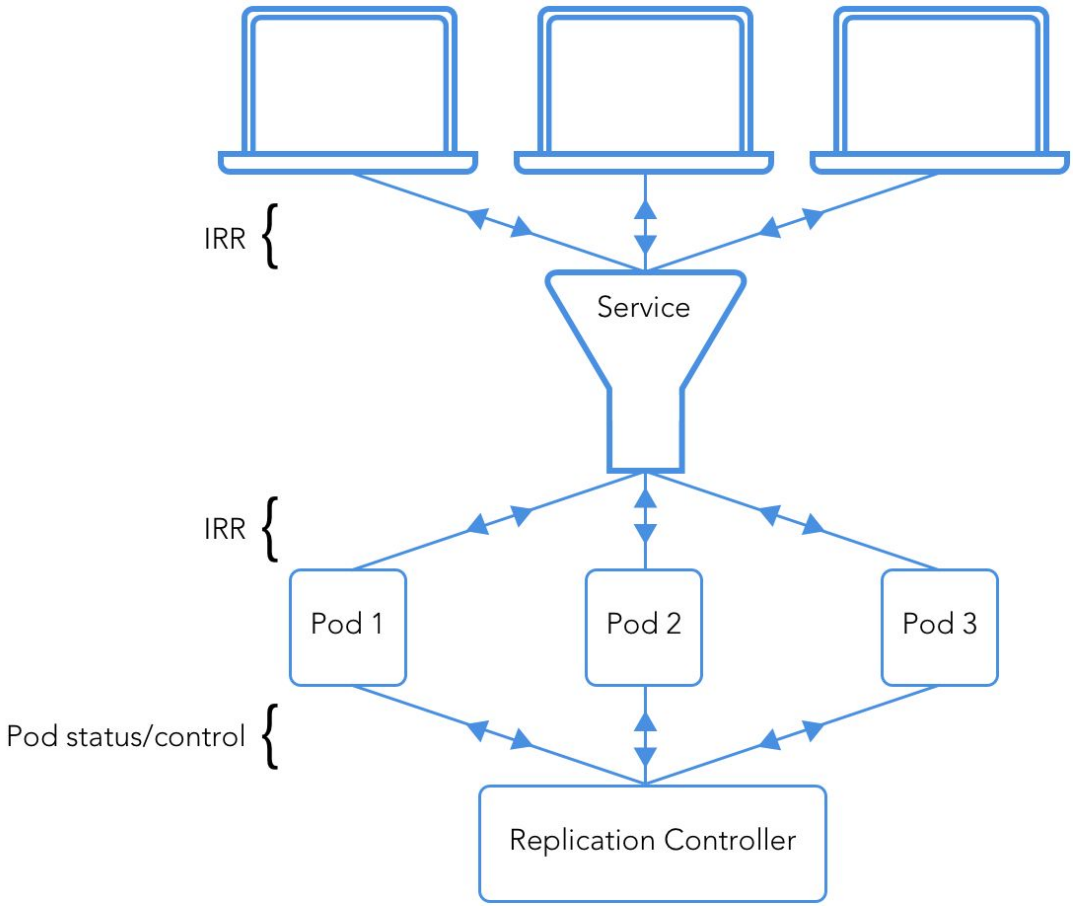
Replication Controllers

A controller for ensuring a given number of replica pods exist.

Services

A single point of load-balancing access for requests to replica pods.

Architecture



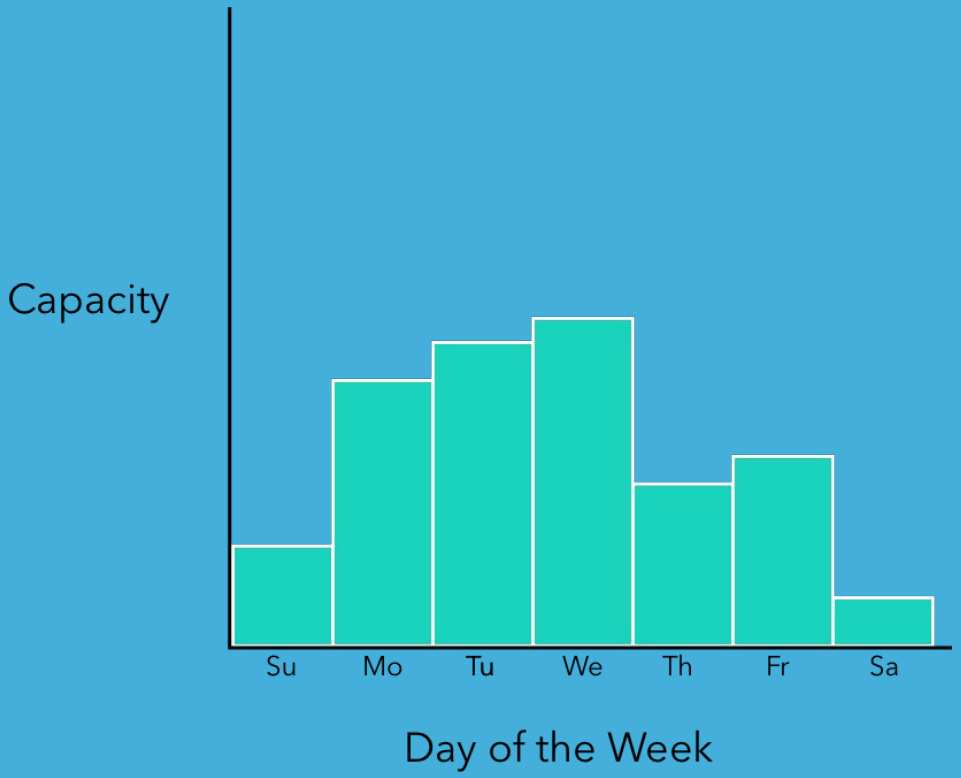
IRR = Individual responses and requests

Accomplishing Specific Goals

How does (predictive) auto-scaling in Kubernetes improve the summation of ERU and QOS?

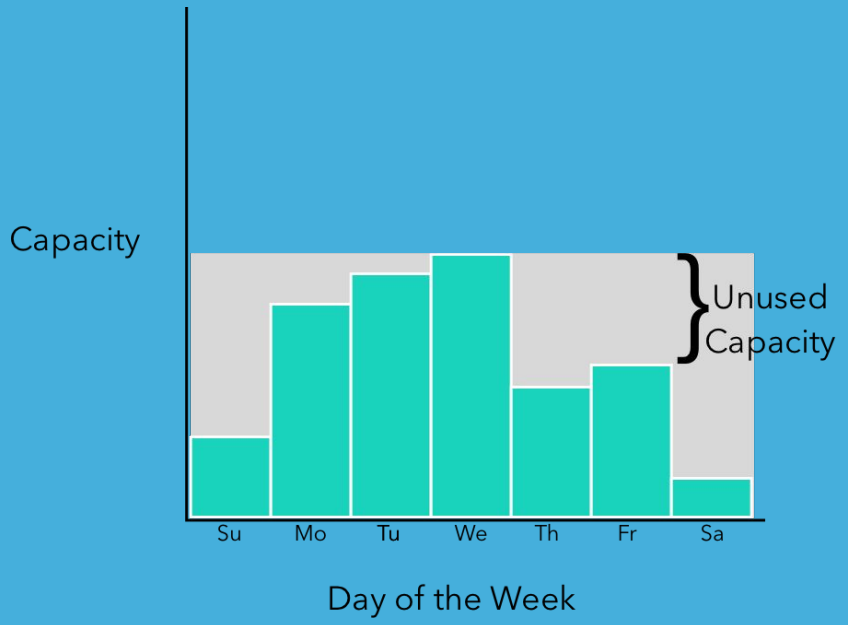
Benefits of auto-scaling

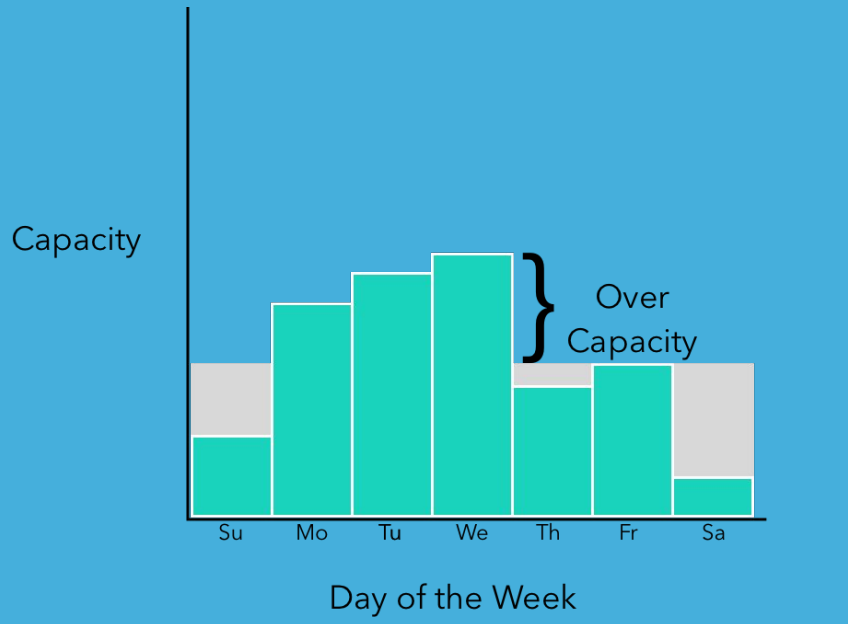
Imagine the following capacity for hbogo.com when running on a cluster manager...



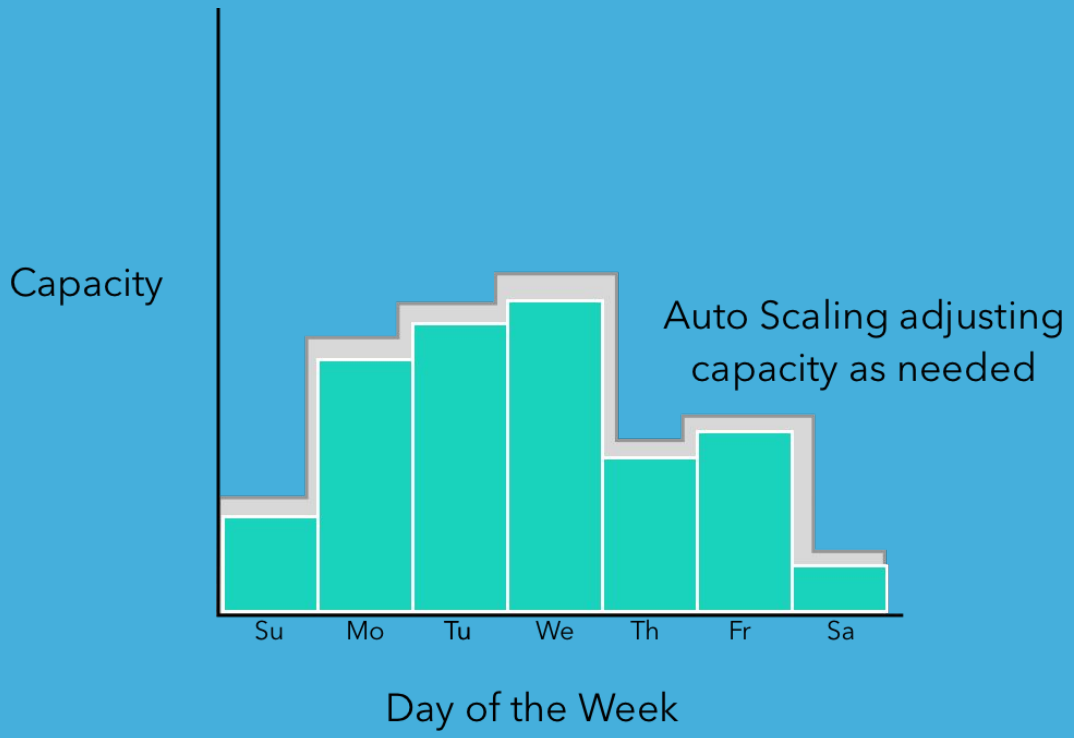
If we do not have auto-scaling

No matter what, either poor QoS or poor ERU.





If we do have auto-scaling



We can assign the application the exact resources it needs, when it needs them... improving the summation of ERU and QOS.

What are the different characteristics of auto-scaling?

Horizontal vs Vertical

How is an application given the extra resources that it needs?

Reactive vs Predictive

Does auto-scaling occur based on the current or future state of the cluster?

What are the major types of auto-scaling?

Threshold-based Rule Policies

Scale if the current resource usage is not in accordance with a set of predefined rules (i.e. Amazon EC2).

Time-series Analysis

Auto-scale based on repeating pattern in the application load (i. e. Netflix).

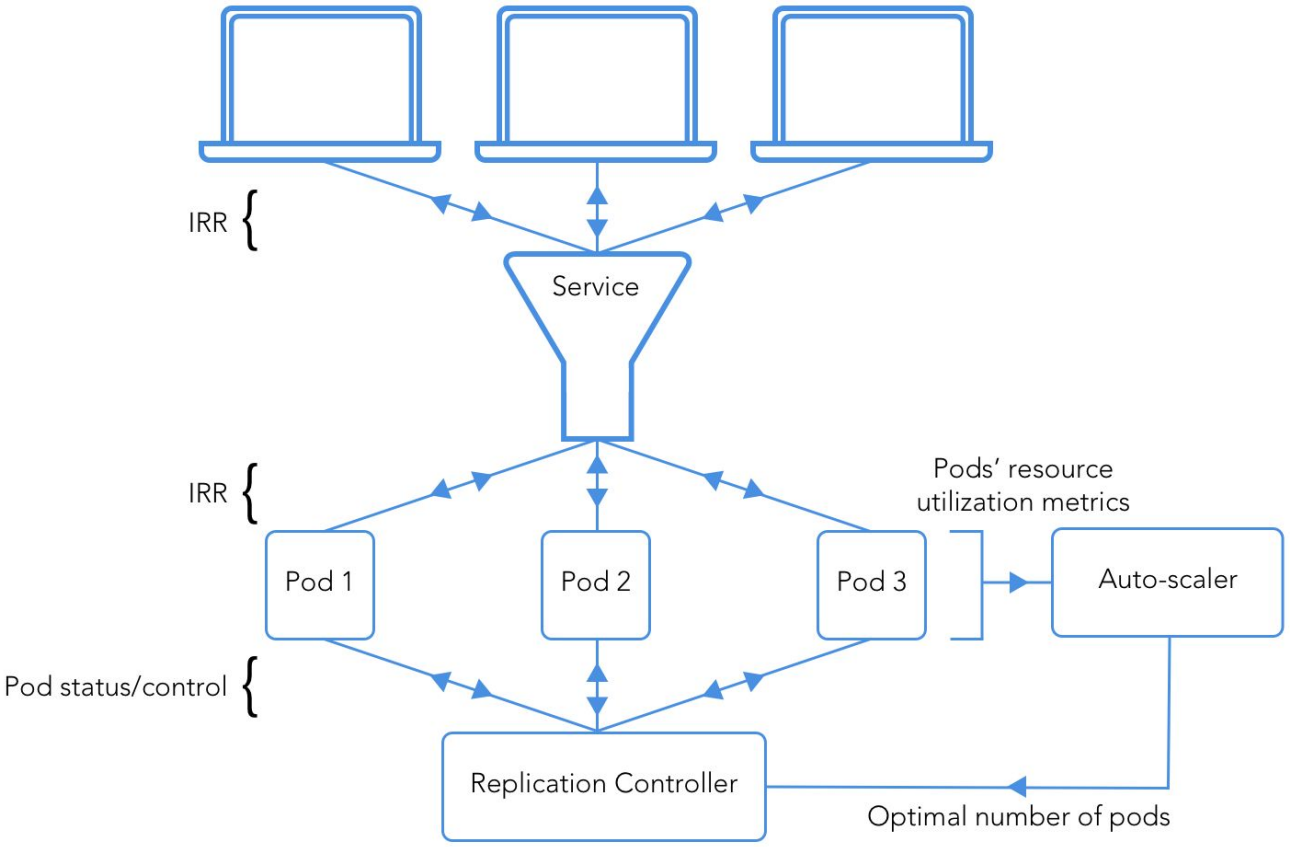
Control-theory (Feedback Control)

Scale such that the resource usage is in accordance with predefined guidelines (i.e. Kubernetes!).

Current State of Auto-scaling in Kubernetes

Kubernetes currently implements reactive, horizontal feedback control based auto-scaling of pods.

Architecture



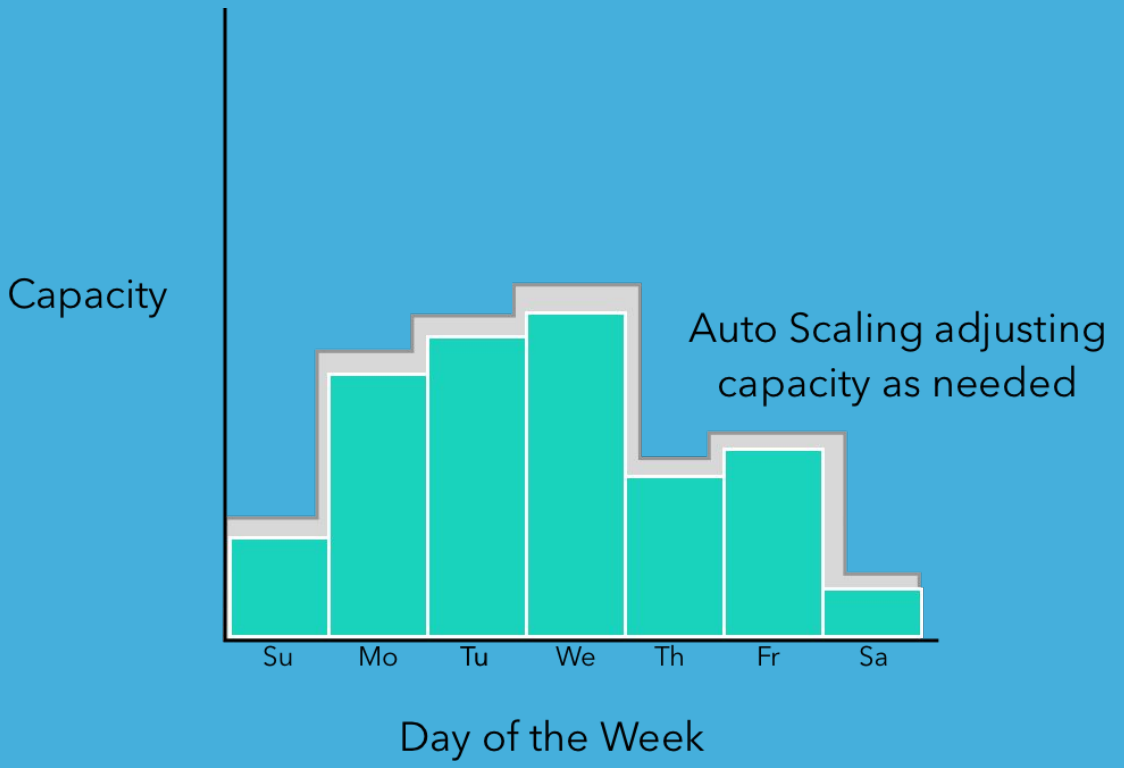
IRR = Individual responses and requests

Concerns with Auto-scaling in Kubernetes

Are there ways to improve the summation of ERU and QOS?

Delayed Pod Initialization Time

What if it takes a long time for a pod to be ready to share in the computational work?



Improvements to Auto-scaling in Kubernetes

What if we add prediction?

Benefits of adding prediction

- Predictive, horizontal feedback control based auto-scaling of pods
- Improves the summation of ERU and QOS

A case study

- Imagine at 9:50pm, hbogo.com needs 100 pods, and at 10pm the season of premier of Silicon Valley is shown, so we need 200 pods.
- Imagine pods take 10 minutes to download the video files they will serve.

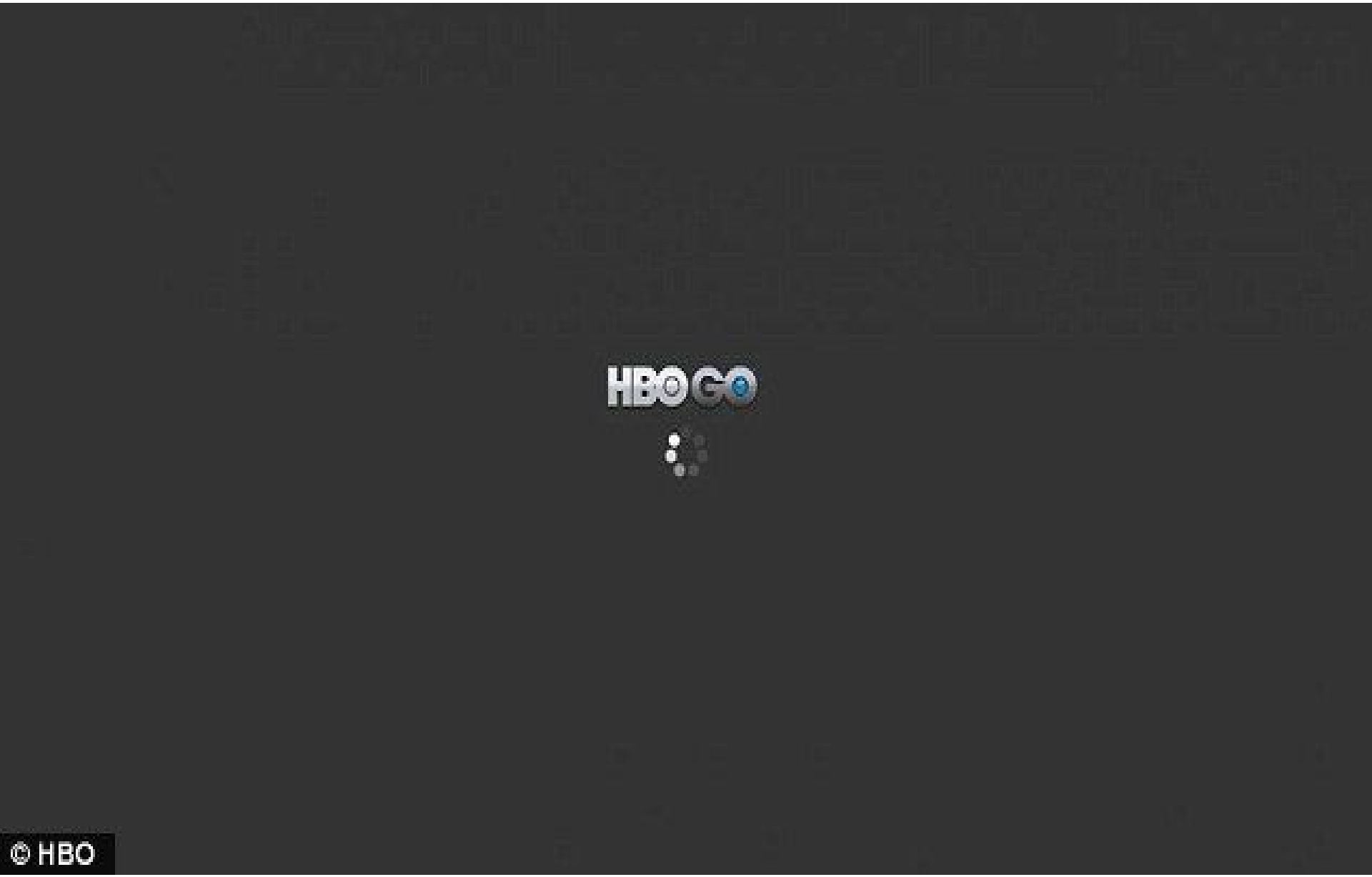
Reactive

1. At 10:00pm, reactive auto-scaling says create 100 pods.

2. From 10:00 to 10:10, wait for pods to initialize.

3. At 10:10, all the needed pods will be working.

For 10 minutes, hbogo.com operates with only half the resources it needs, while we wait for the replica pods to initialize.


HBO GO
© HBO

Predictive

1. At 9:50pm, predictive auto-scaling says create 100 pods.

2. From 9:50 to 10:00, wait for pods to initialize.

3. At 10:00, all the needed pods will be working.

hbogo.com **always** has the resources that it needs.

Implementing Predictive Auto-scaling

How did we actually do this?

Implementation questions?

How long does it take for a pod to be ready to share in the work?

How can we store previous measurements of resource utilization?

How do we auto-scale with that information?

Finding pod initialization time

- Pod initialization time (PIT) is how long it takes the pod to share in the work... NOT how long it takes the container to create.

PIT = (ReadyTimestamp - CreationTimestamp)

Storing previous measurements

- Keep a finite list of tuples of timestamps and CPU utilization percentages for each auto-scaler

[{timestamp_1: cpu_1}, {timestamp_2: cpu_2}, ...]

Auto-scaling predictively

- Use previous observations to calculate a linear line of best fit (X = Time, Y = CPU)

$$b = Cov_{XY}/Var_X$$

$$a = mean(Y) - b * mean(X)$$

$$t = CurrentTime + PIT$$

Auto-scaling predictively con't

- Use line of best fit to predict future resource utilization

$$fru = a + b * t$$

- Use future resource utilization in place of current resource utilization in auto-scaling algorithm

Auto-scaling algorithm

- The number of replica pods that should exist is determined by the following equation:

TargetPods = SumPodsResourceUtilization / TargetResourceUtilization

Put it all together...

And you get predictive auto-scaling!

Evaluation

How will we know if we're successful?

Goals of evaluation

- What is predictive auto-scaling's impact in comparison to reactive auto-scaling?
- When does predictive auto-scaling perform well? When does it not perform well?

What are the metrics we'll use?

How do we measure ERU?

How do we measure QOS?

How do we combine the two?

$$ne_t = ((e_t - MEAN(E_A)) / STDDEV(E_A))$$

$$nq_t = ((q_t - MEAN(Q_A)) / STDDEV(Q_A))$$

$$s_t = -ne_t + -nq_t$$

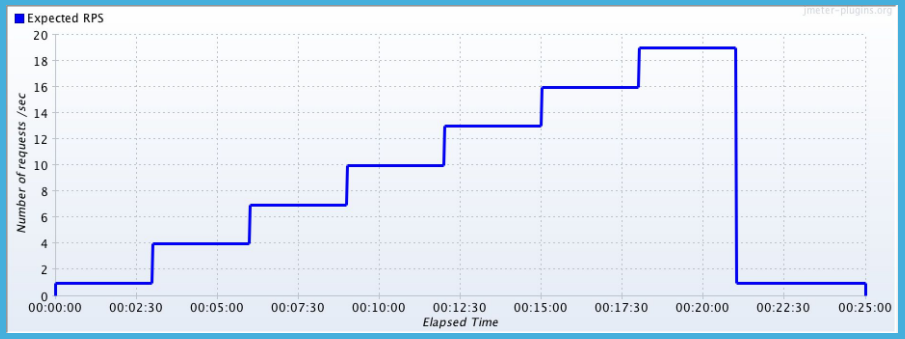
What are the independent variables?

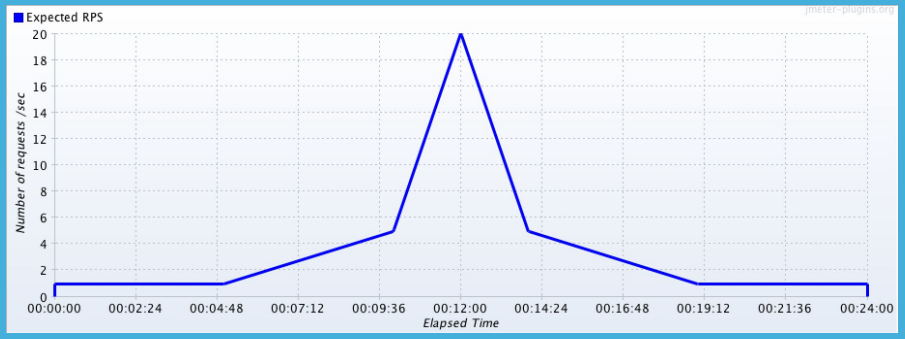
Auto-scaling method (i.e. predictive, reactive, or static)

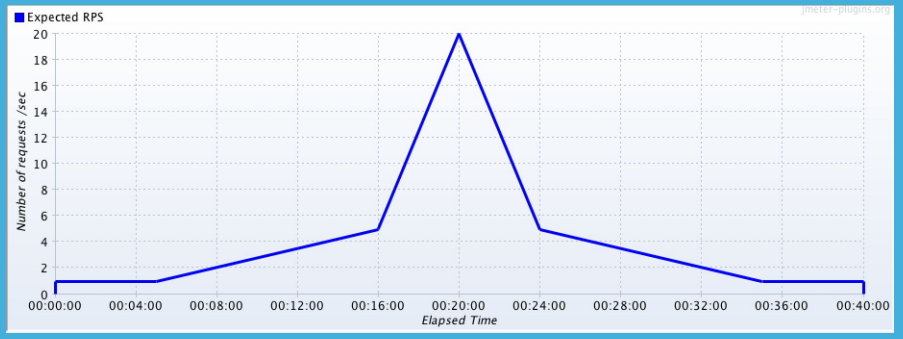
Pod Initialization Time (i.e. 135s vs 5s)

Traffic Request Pattern (i.e. step-ladder, jagged-edge, increase-decrease, flash-crowd)

Traffic request pattern







What tools do we need?

test-server: A custom containerized web server which allows us to control pod initialization time and record metrics.

Jmeter: Allows us to create HTTP traffic following a specific pattern.

InfluxDB: Records our time-series evaluation data.

How do we use these tools?

Deploy test-server and Jmeter onto Kubernetes.

Store the results in InfluxDB.

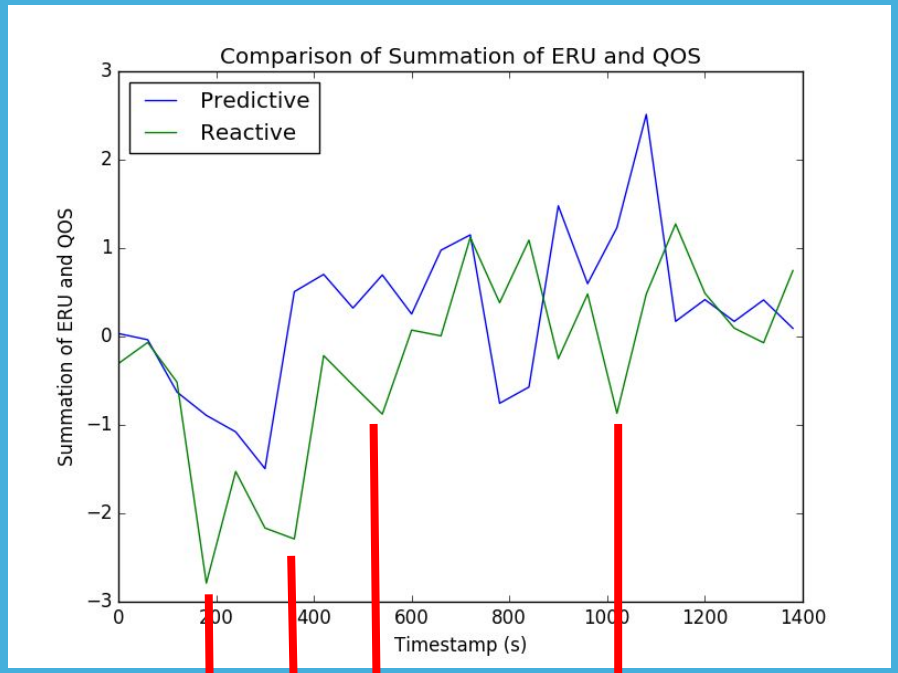
Retrieve the data from InfluxDB, process for summary statistics and graphs.

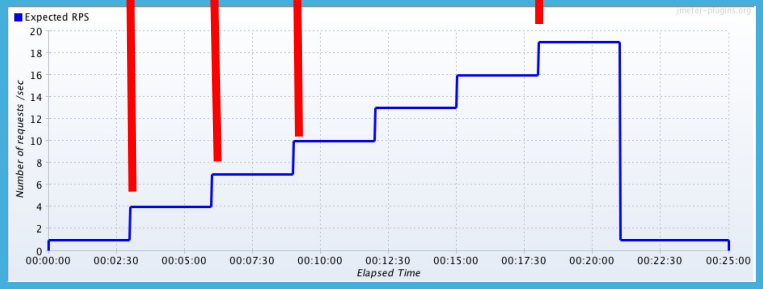
All Automated!

What are the results?

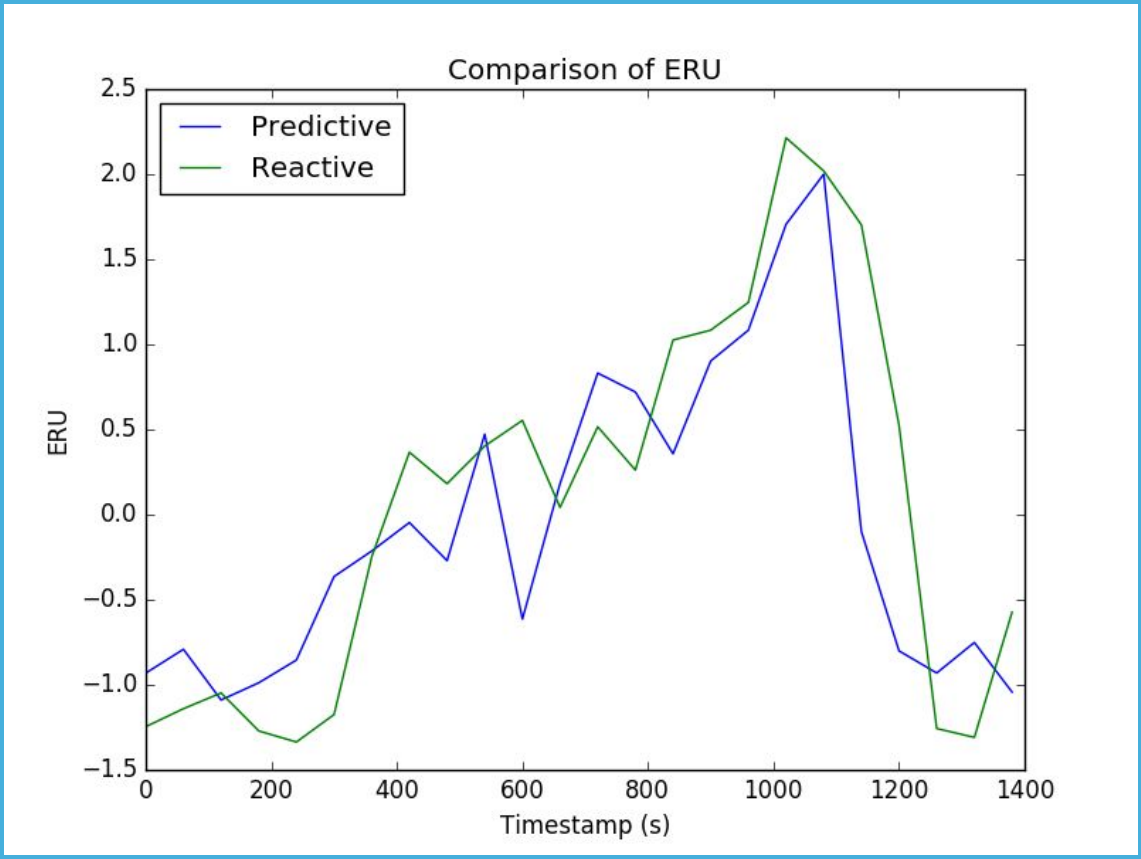
Will I be able to watch my TV shows without interruption??

Results for step-ladder

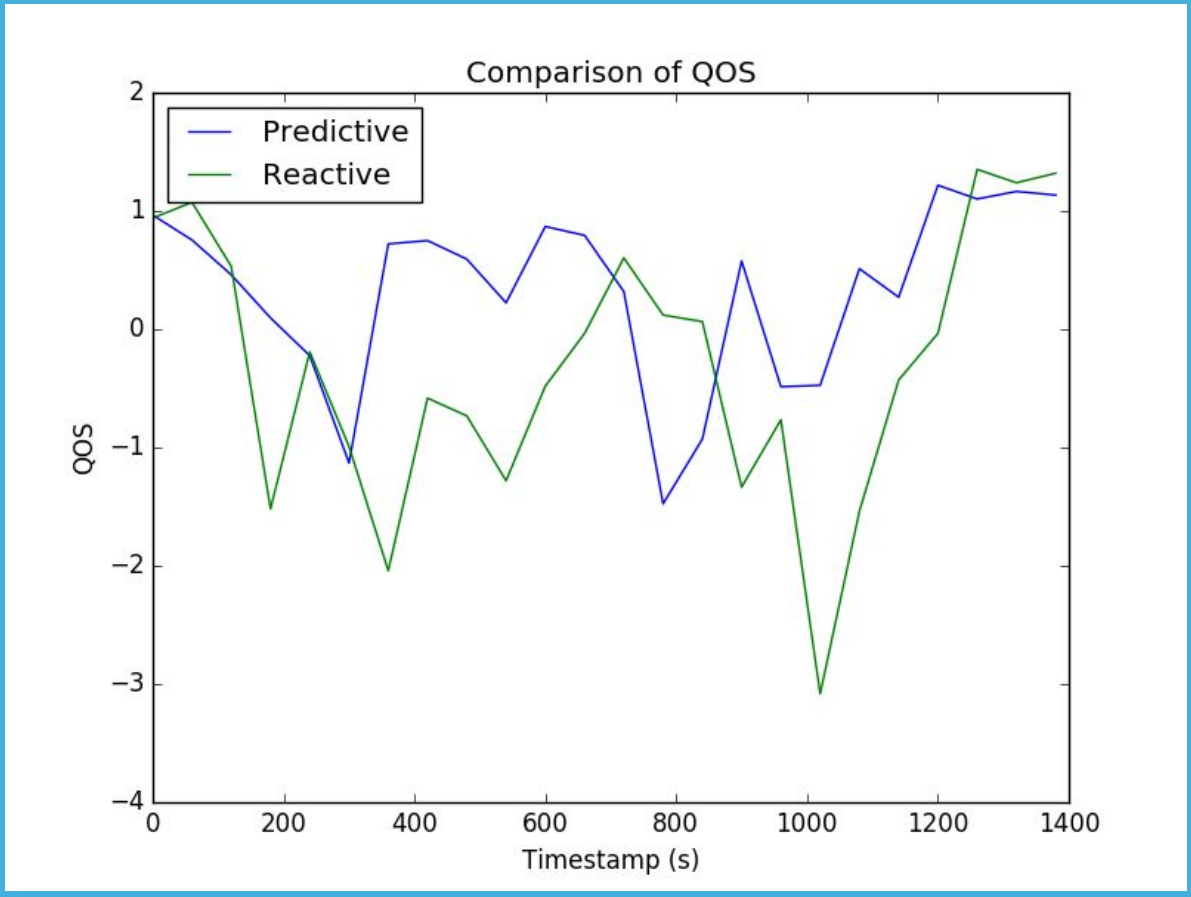




What about just eru?



What about just qos?

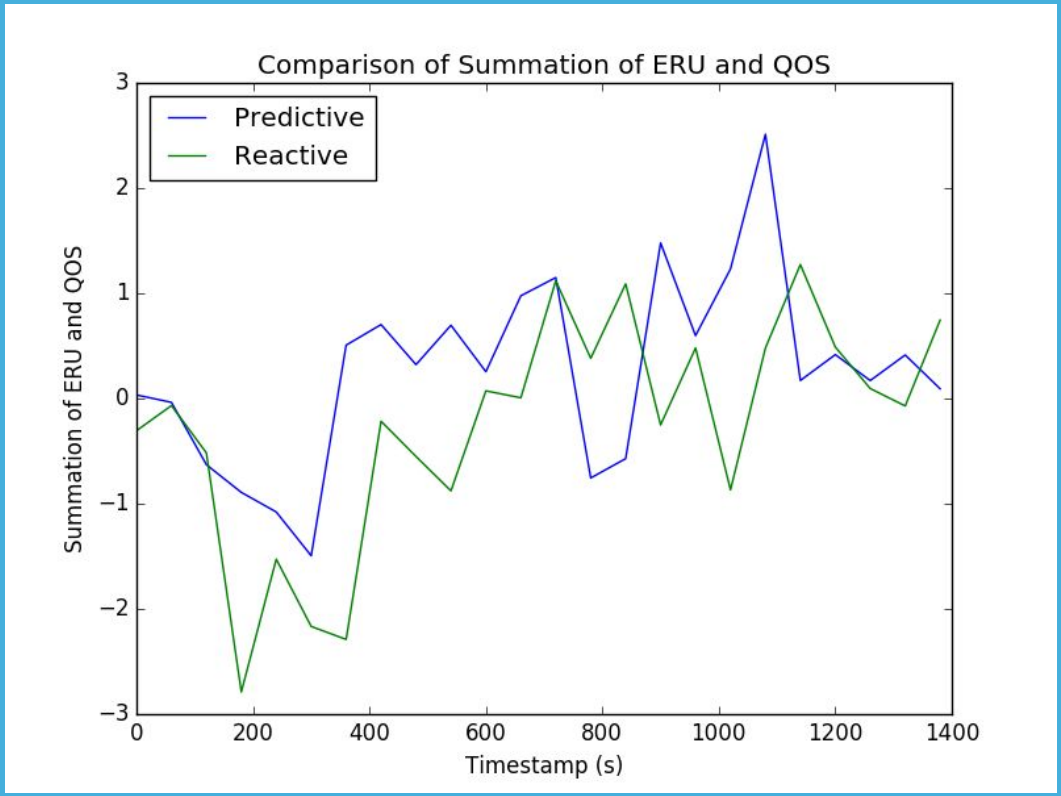


What about statistical significance?

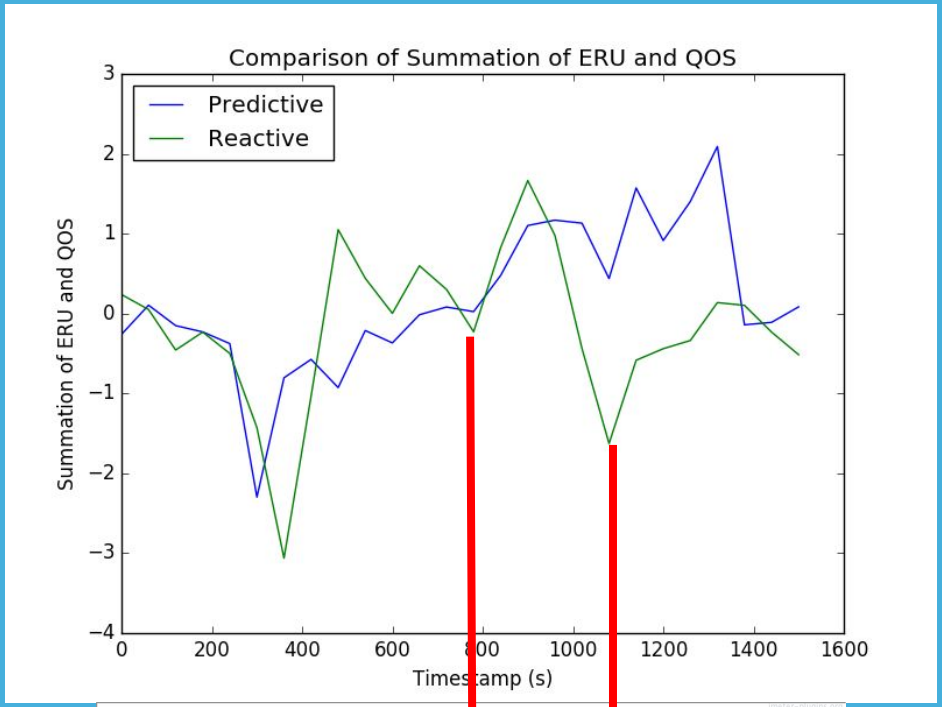
- Our null hypothesis is there is no difference between predictive and reactive horizontal auto-scaling with respect to summation of ERU and QOS at each matching time interval.
- Our alternative hypothesis is predictive auto-scaling has a higher summation of ERU and QOS at each time interval compared to reactive auto-scaling.

What about statistical significance cont'd?

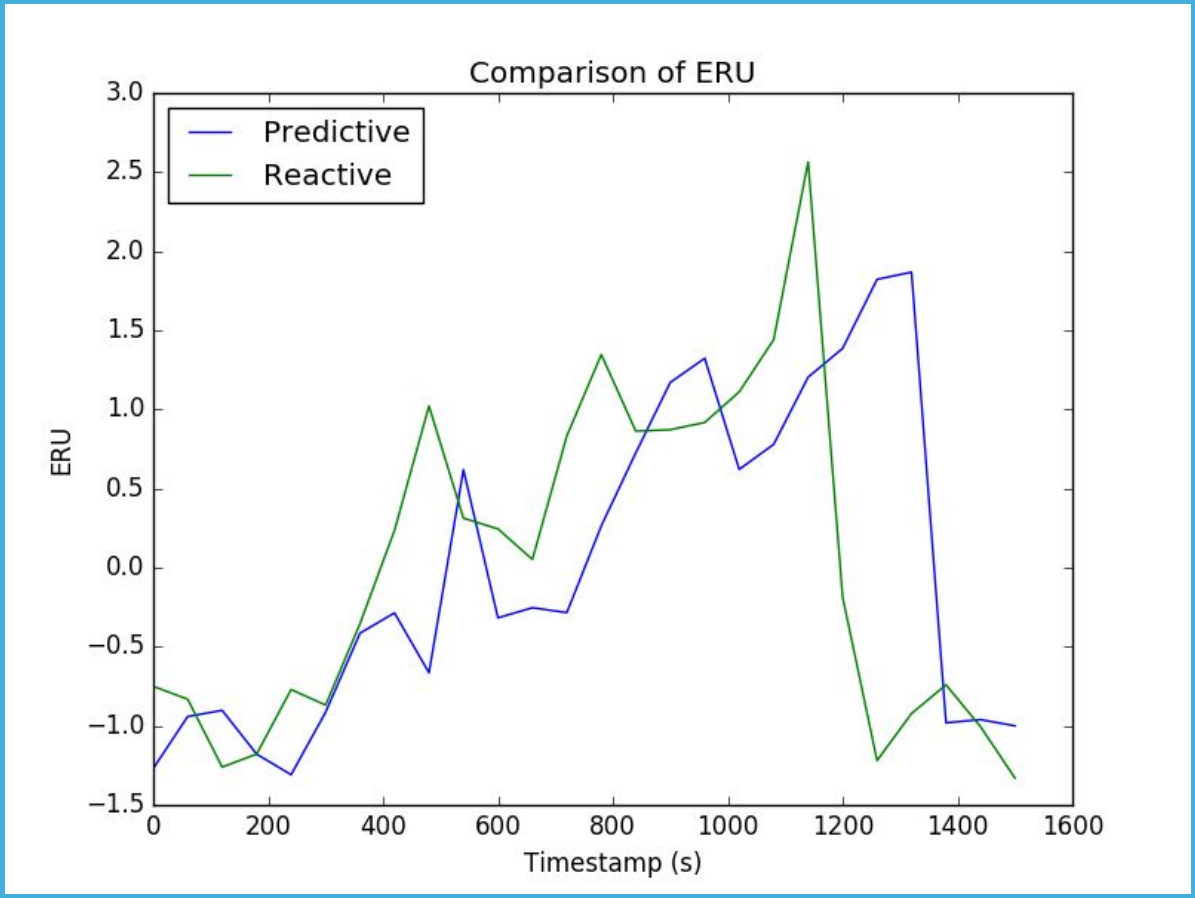
- p-value: .315
- z_score: .482
- mean: .522
- std_dev: 1.084



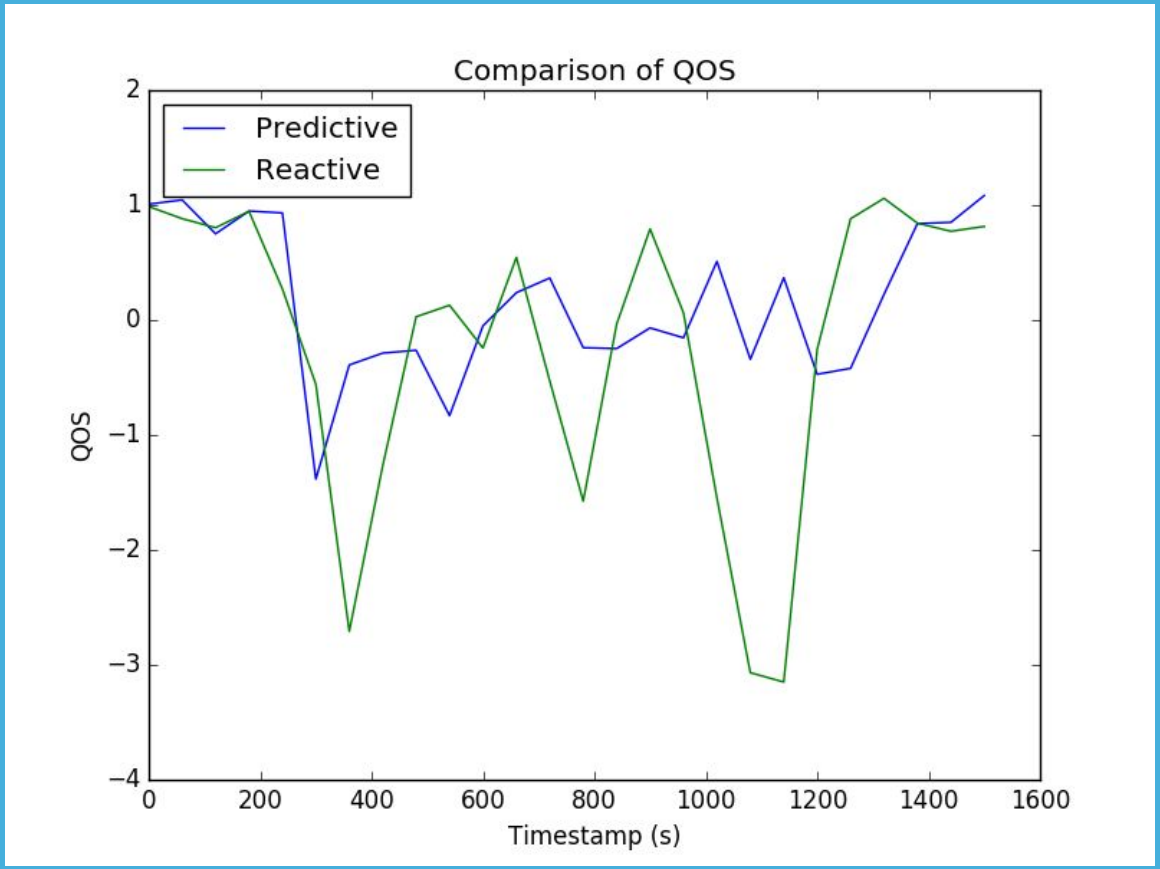
Results for jagged-edge



What about just eru?

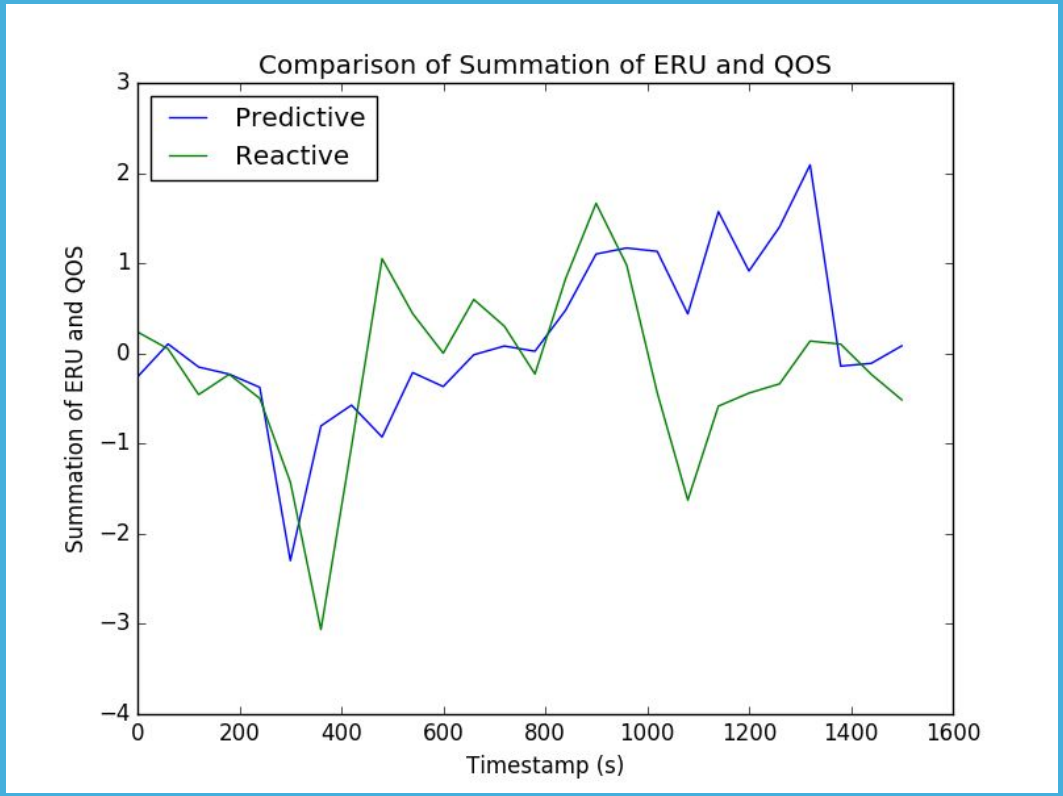


What about just qos?

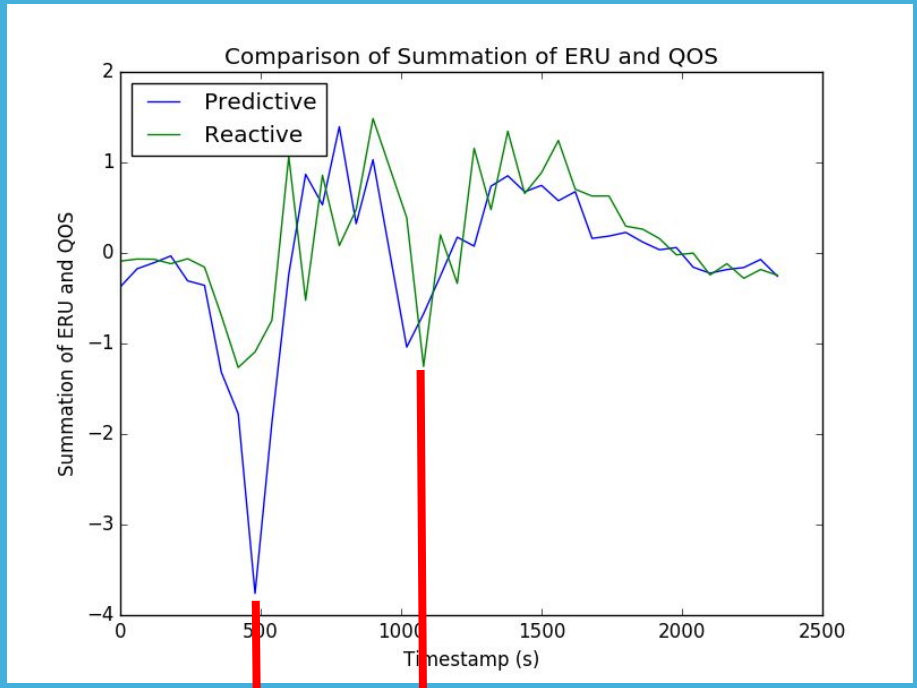


What about statistical significance cont'd?

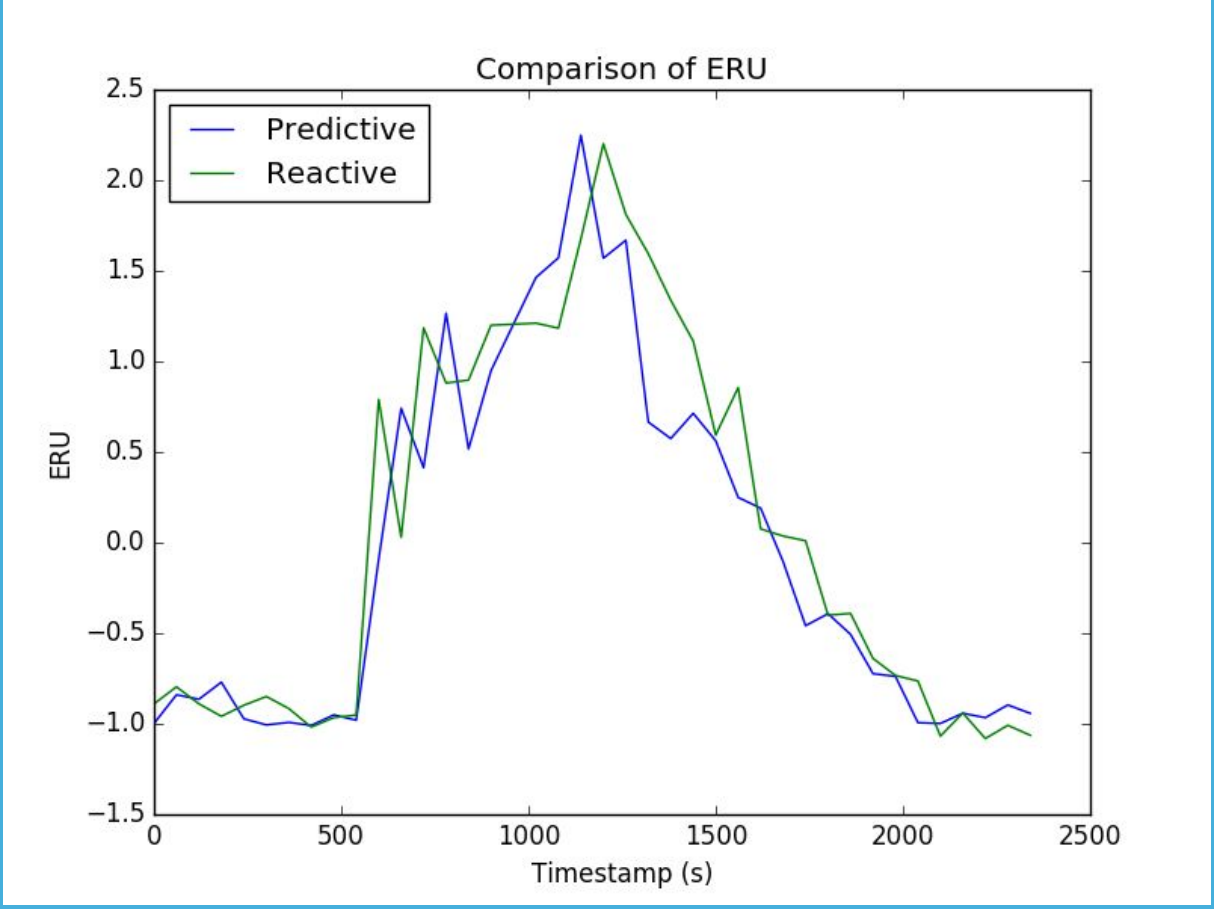
- p-value: .374
- z_score: .321
- mean: .340
- std_dev: 1.062



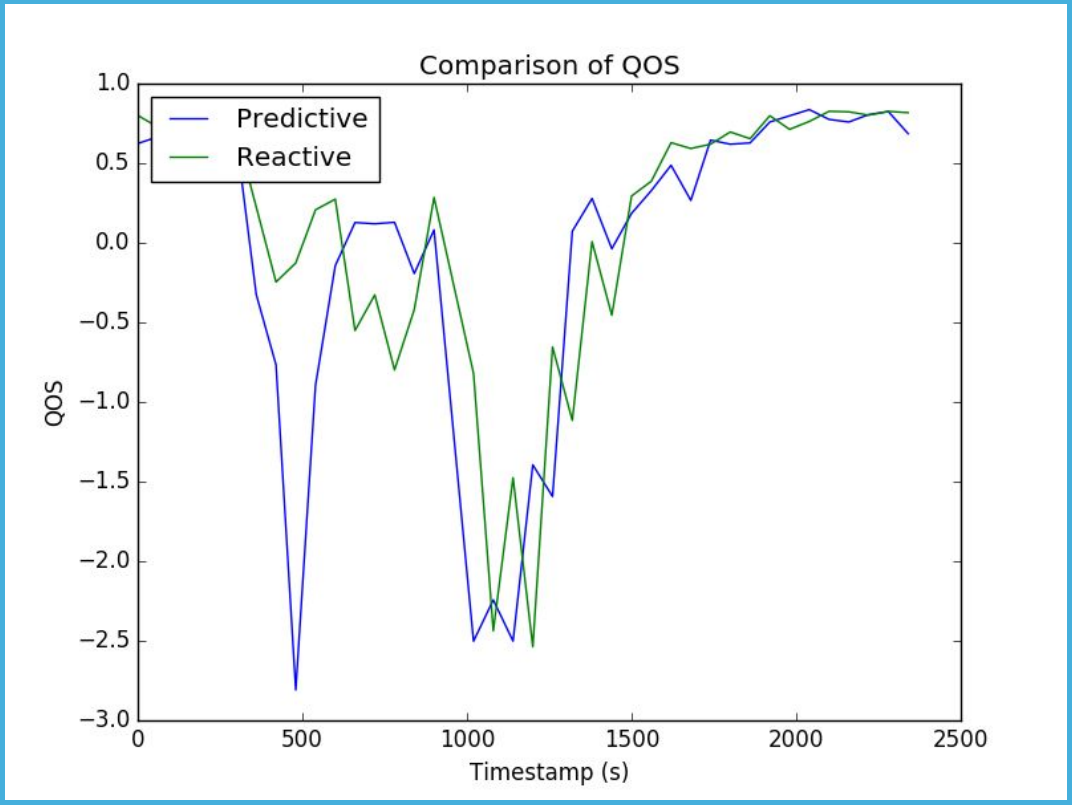
Results for increase-decrease



What about just eru?

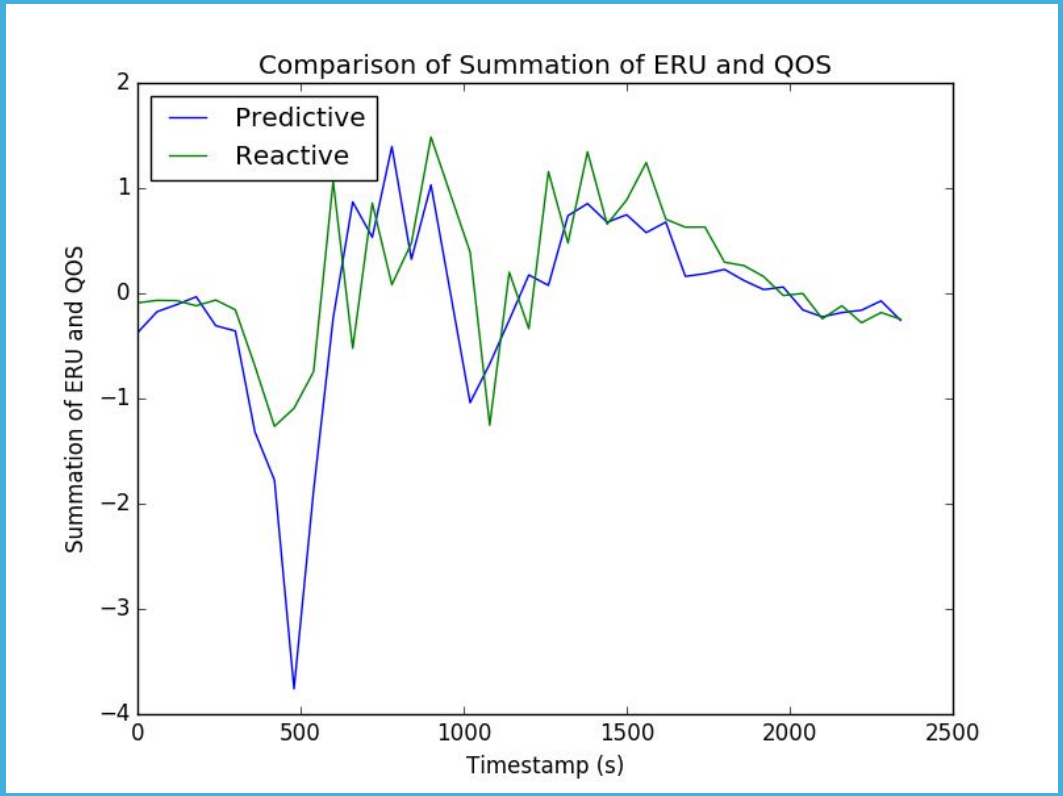


What about just qos?

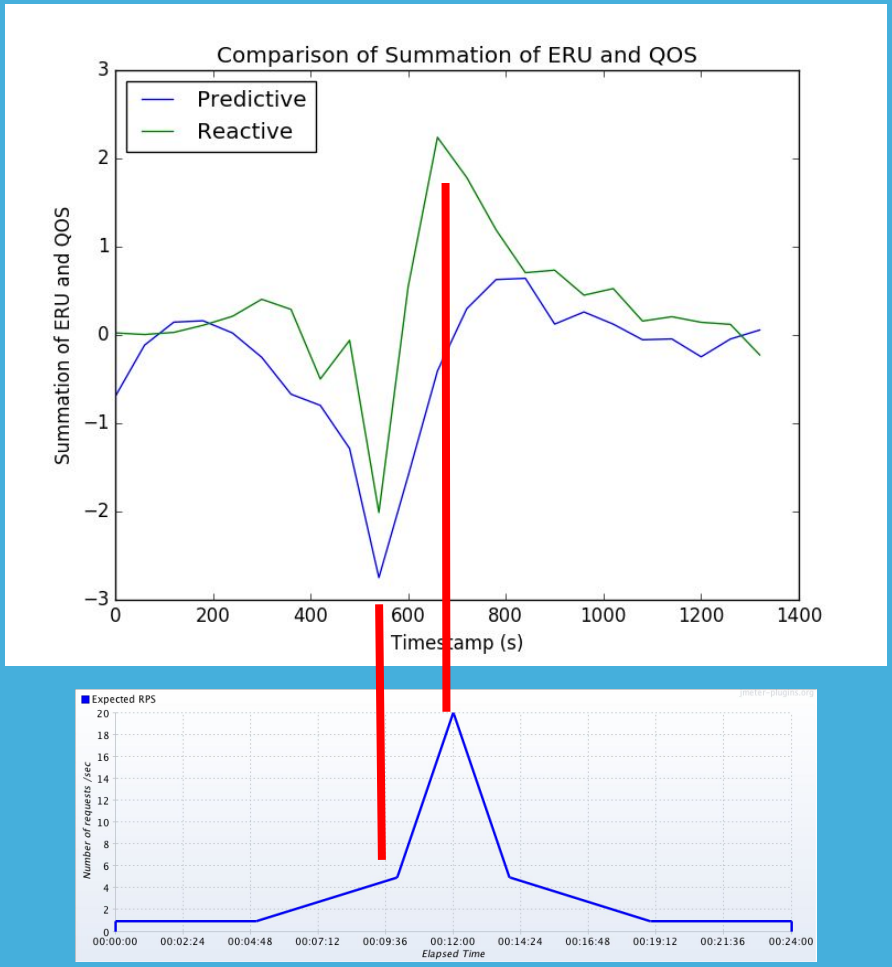


What about statistical significance cont'd?

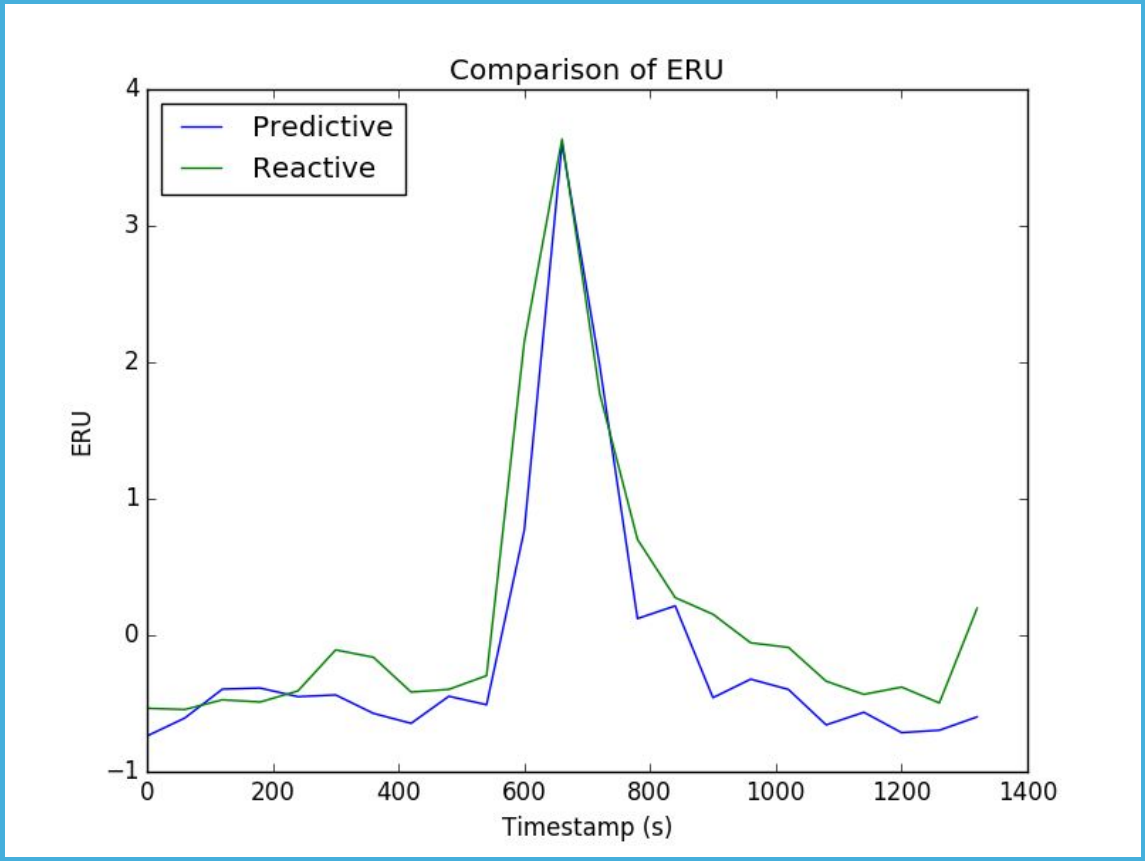
- p-value: .638
- z_score: -.352
- mean: -.239
- std_dev: .680



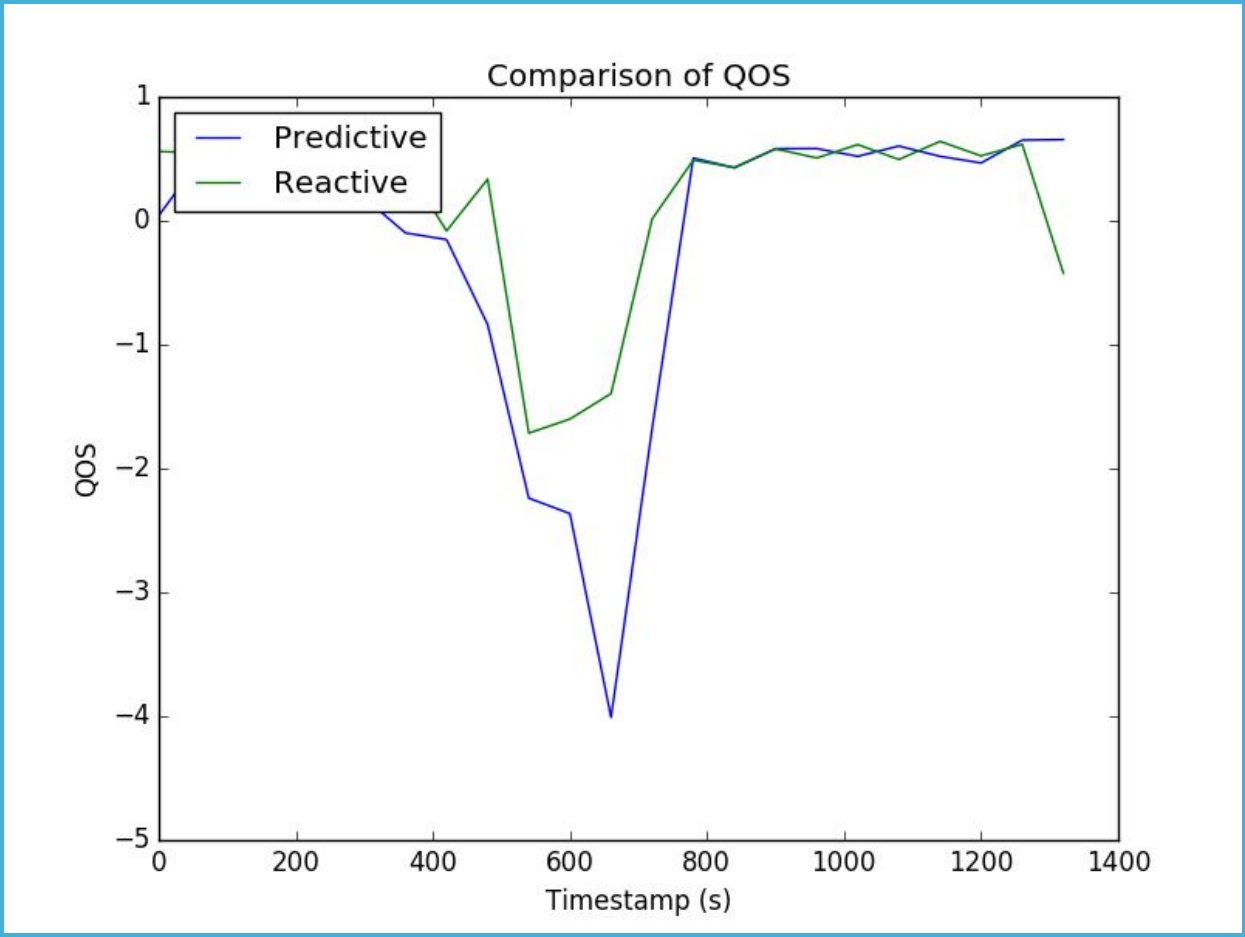
Results for flash-crowd



What about just eru?

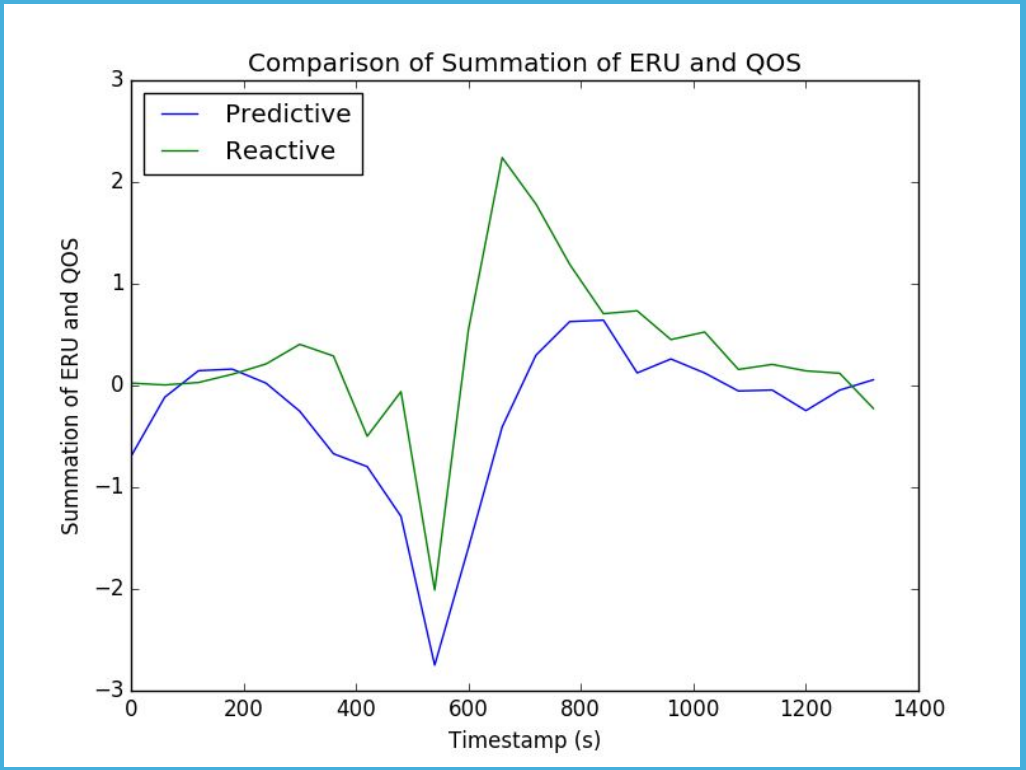


What about just qos?



What about statistical significance cont'd?

- p-value: .802
- z_score: -.850
- mean: -.591
- std_dev: .695



Conclusion

Where we are now and looking to the future?

Summary of Contributions

- Formalized the auto-scaling problem and defined success
- Conceived of and implemented predictive auto-scaling in Kubernetes
- Evaluated predictive auto-scaling

Future work

Evaluate different testing traffic patterns and gather real world data

Different methods of predicting future resource utilization (maybe even Machine Learning!)

Expand to work with custom metrics beyond just CPU and memory

Future work cont'd

Predictive down-scaling, in addition to predictive up-scaling

Determine the optimal size of the forbidden window for re-scaling after a scaling occurred

Merge predictive auto-scaling into the mainstream Kubernetes distribution

Want to help?

https://github.com/mattjmcnaughton/kubernetes

THANKS!

Any questions?

CREDITS and CITATIONS

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash
- Thanks to Andrew Udell for assistance with the graphs and images.
- All Kubernetes info is from http://kubernetes.io/.
- Lorido-Botrá n, T., Miguel-Alonso, J., and Lozano, J. A. Auto-scaling Techniques for Elastic Applications in Cloud Environments. Research EHU-KAT-IK, Department of Computer Architecture and Technology, UPV/EHU, 2012.