Predictive Pod Autoscaling in the Kubernetes Container Cluster Manager

by

Matt McNaughton

Professor Jeannie Albrecht, Advisor

A thesis submitted in partial fulfillment
of the requirements for the
Degree of Bachelor of Arts with Honors
in Computer Science

Williams College
Williamstown, Massachusetts

October 27, 2015

DRAFT

Contents

Abstract

Acknowledgments

Chapter 1

Introduction

1.1 Motivation

The Kubernetes cluster manager [3] controls all aspects of a cluster through admitting, running, and restarting applications, monitoring and displaying application health, and maintaining the underlying machines composing the cluster. Currently the Kubernetes cluster manager implements autoscaling, ensuring applications scale to meet varying external demands. [2] We seek to improve autoscaling to decrease the amount of time replicated applications must handle too much or too little demand, increasing the efficiency and responsiveness of Kubernetes.

1.2 Process and Implementation

Kubernetes is uniquely both completely open source under the Apache License, and in production use at Google. [1] Ultimately, the goal is for the Kubernetes project to accept this work into their master distribution. Thus, the norms and requirements of the Kubernetes community, as well as the current implementation of Kubernetes, will influence the implementation of our autoscaling modifications. The implementation steps are as follows:

- Propose autoscaling modifications to the Kubernetes community.
- Make regularly scheduled pull requests to gradually introduce proposed changes in standalone components.
  - Importantly, Kubernetes is an active project, and it will be important to continuously merge our changes to decrease the cost on ourselves and on the project maintainers of combining two divergent branches.
- Evaluate proposed changes to determine their validity and performance.
- Make necessary changes and extensions.

1.3 Goals

Fairly defined methods and metrics exist for measuring the efficiency of cluster managers. Particularly, we will define a singular metric to measure the success of autoscaling, and then compare the values of this metric when running both statistically simulated and historically recorded data on Kubernetes with, and without, our modification.

Additionally, Kubernetes is a system used to run a variety of different types of applications, in a variety of different environments. Equivalently, our modifications to autoscaling will undoubtedly incorporate many tunable parameters and application-definable variables. An additional goal is determining the use cases in which our modifications are most effective, and just as importantly, the use cases in which they are least effective. In conjunction, after making our modifications we seek to determine a suitable, default behavior for autoscaling.

Part of the appeal of this thesis is the ability to contribute to a system running applications used by millions of people each day. Thus, a final metric of success is the extent to which the method and implementation we use to improve autoscaling conforms to the architecture and standards of Kubernetes, and the extent to which our modifications are accepted by the Kubernetes community.

Chapter 2

Background

2.1 Cluster Management

2.1.1 History and Motivation

2.1.2 Alternatives to Kubernetes

Borg

Omega

Others

2.1.3 Kubernetes

2.2 Containerization

2.3 Prediction Models

Bibliography

[1] Google container engine. https://cloud.google.com/container-engine/docs/.

[2] Kubernetes horizontal pod autoscaler proposal. https://github.com/kubernetes/kubernetes/blob/master/docs/proposals/pod-autoscaler.md.

[3] Kubernetes website. http://kubernetes.io.