Predictive Autoscaling of Pods in the Kubernetes Container Cluster Manager

Matthew McNaughton
Advisor Jeannie Albrecht
Computer Science Thesis, Williams College

October 20, 2015

1 Motivation

Society is becoming increasingly dependent on computation. Every facet of human life, including health care, finance, and social interactions now involves computers, resulting in a boom in the demand for computational power. This explosion in need, which shows little signs of abating, drives an interest in cluster computing.

Cluster computing combines the resources of thousands of commodity computers to create the mass of computing power necessary to serve high traffic sites, process massive MapReduce jobs, etc... While the biggest firms in industry have used computing clusters, and the requisite cluster managers, for approximately the last decade, this incredible computing power is increasingly becoming available to the general public. Kubernetes, an open source cluster manager from Google, leads this effort. [4]

Kubernetes manages many aspects of the cluster, including running processes, maintaining machine health, and providing simple methods of querying the cluster state. Kubernetes runs similar processes in a unit of computing called a pod. Pods have various performance requirements and face ever changing external factors (ie. a variable number of server requests). Given both these concerns, Kubernetes implements horizontal pod autoscaling, through pod replication. [3] Kubernetes autoscales pods to ensure each pod honors a given threshold metric (ie. max 80% CPU usage). Replicating pods essentially duplicates the number of processes being run, allowing a division of labor among computing resources and decreasing the individual load on any specific pod. For example, if a single pod was responsible for all web requests, replicating another pod would allow each pod to now handle half the requests. With horizontal pod autoscaling, a computational metric functions as a threshold which, if surpassed, triggers pod replication until the metric returns below the threshold value.

Introducing horizontal pod autoscaling greatly increases the responsiveness of Kubernetes to a variance in external environment. However, it is possible to make Kubernetes even more responsive. Specifically, the current method of autoscaling does not account for the time necessary to replicate pods. For example, if creation time for a pod is five minutes, even if the autoscaler triggers replication as soon as the metric threshold is crossed, it is still necessary to wait for the pod to be created before the work can effectively be shared. This latency becomes especially problematic when the costs of the pod being past the threshold are exceptionally high, pod replication takes a non-trivial amount of time, or the demands on pods are particularly variable. Clearly, while horizontal pod autoscaling offers considerable benefits to the Kubernetes cluster manager, there are still steps to be taken to address the previously listed issues.

2 Question

In this thesis, I will seek to address the deficiencies previewed in the motivation section. Specifically, I will answer the following question: Given a pod computation threshold metric triggering autoscaling, is it possible to decrease the amount of time spent above the metric's threshold, without substantially increasing the net overall pod runtime? In other words, is it possible to predict when the threshold metric will be crossed, so the replicated pod becomes available exactly when needed?

3 Hypothesis

I hypothesize the answer to the stated question is yes. Confidence in this hypothesis derives from the possibility of implementing preemptive scaling. [1] Preemptive scaling measures the rate of change for the given pod threshold metric, and predicts when the threshold will be crossed. Combined with knowledge of the replication time for the pod, the creation process for the replica can begin before the threshold is ever crossed. When preemptive scaling works, as I hypothesize it will, the replication pod is available exactly when needed, and the initial pod must spend minimal time operating above the threshold metric while waiting for the pod replication to complete.

4 Implementation and Methodology

Kubernetes is a rare and interesting project in that it is both completely open source and in production use at Google and many other companies. [2] Thus, I will be implementing my work with preemptive autoscaling on the actual Kubernetes code, with the goal of my work being accepted into the master branch, as part of the main Kubernetes distribution.

A fairly defined methodology exists for evaluating cluster managers like Kubernetes. [5, pg. 355] The first aspect of evaluation utilizes a lightweight simulator based on the statistical structure of the data I wish to measure. For example, if I wanted to test the ability of Kubernetes with preemptive autoscaling to respond to a spike in web traffic to a given pod, I would simulate the highly variable incoming web requests with statistical modelling. The second evaluation replays historic data, in order to make exact conclusions about how Kubernetes with preemptive autoscaling would have operated in a specific situation. This analysis is typically more restrictive, because it can be difficult, and time-consuming, to obtain and replay exact data.

Evaluation will involve tracking the internal metrics Kubernetes already records, including the amount of time a pod is running, and specific computational metrics for the pod (ie. CPU, memory. . .). Some work will be necessary to decide which metrics are best for measuring scaling responsiveness.

5 Current State

Working with Kubernetes has been extremely encouraging. Most importantly, the lead developer of Kubernetes, Brendan Burns '98, is a Williams graduate who has been very generous with his time and expertise, as we have discussed potential areas of exploration. I am hopeful that I will be able to continue working with Brendan going forward.

Naturally, Kubernetes is a large open source project, and as such there are certain standards and customs which must be followed. However, I have contributed to open source for over a year, and am familiar with the workflow; I have already used my previous experience to make small contributions to Kubernetes. I am optimistic and excited about continuing to work with Kubernetes and the code I write for my thesis hopefully being accepted into the main branch of the project.

References

[1] Conversation with brendan burns, lead developer of kubernetes.

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

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

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

[5] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 351–364, Prague, Czech Republic, 2013.