Temporary Thesis

**Introduction**

Starting a web-based company is a very risky endeavor. Recent statistics place the startup failure rate above 90% for varying reasons [1]. This statistic should be frightening to many who are trying to begin their own business, especially as the cost of hardware for such an endeavor has traditionally costed in the thousands of dollars before the service even launches. This, however, is the problem that the Cloud market specifically aims to rectify.

The Cloud, as it is today, is a collection of hardware resources which are owned by several large, established companies and rented out for use by other companies. This market takes many forms, such Google’s Compute Engine and Amazon AWS, but all essentially the same: users who wish to use these resources as a platform for their own service, or for whatever needs they may have, pay for the resources they want and are charged by the amount of time they use these resources. As with any resource someone wishes to buy, when they rent this hardware they are guaranteed to receive it and it cannot be taken away from them until they give it up freely. However, since users of Cloud services ask for the exact resource which they want, the providers of said services have very little control over which of their own servers they need to turn on and how much of these servers is actually being utilized for a given user.

For example, consider the case where only one user among thousands asks Google for their highest speed server type, but they only wish to utilize 10% of this server. Google must now turn on this server, and while it is only at 10% utilization, its power consumption is [NEED SOURCE] very high, most likely well over 50%. This leads Google to a situation where they are paying more to sell this resource than they are to receiving from its use. Seeing this very clear issue, Amazon came up with a solution they called Amazon EC2 Spot Instances [2]. Spot Instances are Cloud computing resources that make up the unutilized space on currently active Amazon servers and are sold to the highest bidders attempting to utilize this resource for less than the standard rate of a Cloud resource. The catch to these is that they are not guaranteed as available to any user for any amount of time and thus can be taken away as soon as someone is willing to pay more for the resource or when the server is no longer needed for on-demand customers.

These spot instances, and their equivalents at other companies such as Google, have provided a unique opportunity. Companies seeking to minimize costs have the option to “take a chance” on these unreliable resources and utilize them as opposed to on-demand servers at full price. While on the surface this seems like a risky endeavor, especially for new companies who can’t risk intermittent service outages while trying to establish themselves, there appear to be ways to avoid losing service while still benefitting from these affordable prices. It is my hypothesis that by gauging the traffic of a service built on unreliable cloud resources we can manage the amount of cloud instances needed to provide 99.99% reliability of a service for the minimum cost available.

To demonstrate this, I intend to create a basic web service and implement it on the cloud using Amazon Spot Instances or Google Pre-emptible Virtual Machines, which act as their version of Amazon’s Spot Instances. This decision is being influenced by numerous factors. Amazon has a much more established spot market, providing very useful past data about their various spot markets that can be used to accurately predict the likely future prices and make the most informed purchasing decisions. However, the Amazon Spot Market is far more complex and utilizes a bidding system which adds to the difficulty in making a perfect algorithm. Google’s Pre-emptible VMs offer a fixed price and a much more fluid integration with Kubernetes, a service that will be utilized to manage server instances. They are also, however, newer and lack the history data of revocations that would be useful in making confident purchasing decisions and they put an absolute limit of 24 hours on any pre-emptible instance they sell. One major factor in deciding is determining how well Kubernetes will be able to manage Amazon Spot Instances across many markets simultaneously.

This web service will be managed by a 2-level management algorithm. The first level acts as a traffic analyzer to estimate the current load on each of the available servers and determine when it is necessary to obtain more resources to maintain reliability and when it is acceptable to drop superfluous resources. When it is decided that the number of resources must change, it advises the second level to respond accordingly. The second level management is the implementation of Kubernetes to manage the Cloud resources. Kubernetes is a very new, open-source project managed by Google which is used to manage server resources across many servers in as efficient a manner as possible. Kubernetes creates containers, which are sealed application packages that exist as seemingly separated portions of a server resource. It then organizes these containers into pods, which are groups of containers that work together to achieve some goal, such as acting as a database for a web service. Different types of pods can then work together to create the service we are trying to achieve. As an example, consider 2 types of pods, one is the front end to the web service and the other is the backend. These can work together to act as a fully operational service.

Kubernetes utilizes a Replicated State Machine approach to providing web services. This means that Kubernetes creates a certain number of copies of a service and directs users wishing to access the service to the least utilized instance of the service. The number of copies of a service is called the desired state and is a setting within Kubernetes. When something happens to change the number of available services, such as one instance of the cloud that was available to Kubernetes previously failing or being revoked, Kubernetes will see some pod has failed and will bring an identical one up in its place. To create an identical pod, Kubernetes references the label attached to the pod that has failed and accesses a template of that pod to create the next instance. The desired state is the parameter being managed by the first level of management, which means Kubernetes will bring up and close down resources as it is instructed by the first level of management.

[1] <https://s3.amazonaws.com/startupcompass-public/StartupGenomeReport2_Why_Startups_Fail_v2.pdf>

[2] <https://aws.amazon.com/ec2/spot/>

[3]