GeoScale: Providing Geo-Elasticity in Distributed Clouds

Tian Guo Prashant Shenoy
College of Information and Computer Sciences
University of Massachusetts Amherst
{*tian,shenoy*}*@cs.umass.edu*

Hakan Hacigümüş†
NEC Laboratories America
Cupertino CA, USA
hakan@nec-labs.com

***Abstract*—Distributed cloud platforms are well suited for serving a geographically diverse user base. However traditional cloud provisioning mechanisms that make local scaling decisions are not well suited for *temporal* and *spatial* workload fluctuations seen by modern web applications. In this paper, we argue the need of geo-elasticity and present GeoScale, a system to provide geo-elasticity in distributed clouds. We describe GeoScale's model-driven proactive provisioning approach and conduct an initial evaluation of GeoScale on Amazon's distributed EC2 cloud. Our results show up to 31% improvement in the 95^{th} percentile response time when compared to traditional elasticity techniques.**

***Keywords*-Distributed Clouds; Resource Management**

I. Introduction

Today's cloud platforms provide a key benefit, in the form of *elasticity*, to help applications handle a time-varying workload. However, cloud applications that service a geographically diverse user base not only experience temporal, but also *spatial* workload variations. Such spatial variations can be caused by reasons such as applications are more popular in one region (e.g. country) than another or regional events (e.g., local festivals or local news stories). Unfortunately, due to this new spatial dynamics, traditional dynamic provisioning techniques that focus on providing elasticity within a single data center are not sufficient. Concurrently, cloud platforms are becoming increasingly distributed and offer a choice of multiple geographic sites and data centers. Therefore, we propose to implement a *geographic-aware* elasticity mechanism (geo-elasticity) in distributed cloud platforms to meet the resource needs of above applications with *geo-dynamic* workloads.

Towards this end, we design a system called GeoScale that enables distributed clouds to autonomously vary the cloud locations as well as the number of servers at each location to handle *both* temporal and spatial variations in application workload. In designing and implementing GeoScale, our paper makes the following three contributions.

- We identify and outline three challenges in designing geo-elasticity for distributed cloud platforms. Essentially, GeoScale needs to monitor and obtain global client workload distributions and provision server resources in suitable set of cloud locations by taking into account the workload intensity and server capability.
- We present a new geo-elasticity technique that can handle dynamics in both the volume and geographic distribution of application workloads. At the core of our approach is a queuing-theoretic model that is seeded with empirical measurements to determine the server capacity needed at each cloud location.
- We conduct an initial evaluation of GeoScale by running representative applications on Amazon distributed cloud platforms. Our experimental results show 13% to 40% improvement in the 95^{th} percentile response time when compared to traditional elasticity techniques.

This work was supported by NFS grants #1229059, #1345300 and #1422245; and a gift from NEC Labs.

† The author is currently affiliated with Google.

II. Background and Problem Statement

A. Background

Our work assumes an Infrastructure-as-a-Service (IaaS) public cloud that comprises data centers from different locations and allows customers to rent virtualized resources in the form of virtual machines (VMs). Customers do not have direct access to the underlying hypervisor on a physical server and must manage their VMs through the cloud's APIs, such as start and terminate servers at a specific location.

Our work targets replicable multi-tier web-based applications that service a distributed diverse client base. Each VM replica hosts either front or back tier or both and we assume applications maintain data consistency of backend replicas using any methods that suits its need [1], [2]. In addition, application providers are assumed to specify performance metrics of interests such as Service Level Agreement (SLA) and request rates. SLA is usually specified in the form of high percentile response time and is used for deriving server capacity in proactive provisioning; while request rates can be monitored for any threshold violations in reactive provisioning.

B. GeoScale Overview

Figure 1 depicts GeoScale's architecture that breaks down into three key components based on their functionalities. Currently, we design GeoScale as a middleware layer that uses public cloud APIs to programmatically manage virtualized resources on behalf of cloud applications. However, our

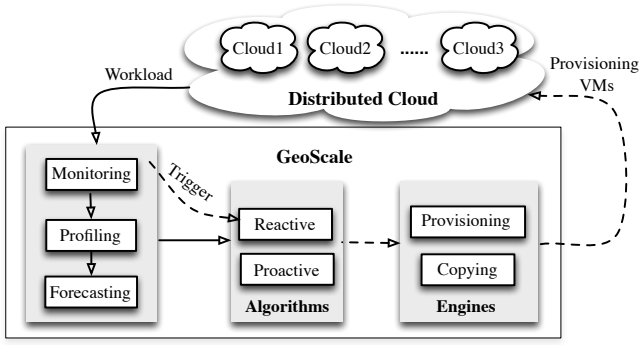

Figure 1: GeoScale Design and Basic Operation.

proposed techniques are easily integrated into cloud platform fabric. The workload engine is responsible for monitoring the incoming client requests, creating a geographic profile of the workload distribution, and employing forecasting techniques such as time-series forecasting to predict future workload based on recent history. GeoScale supports both proactive and reactive provisioning algorithms to handle long-term variations based on predictions and short-term dynamics including workload spikes and forecast errors. Specifically, our proactive algorithm uses queueing-models to provision sufficient resource to meet application-specific SLA. Finally, provisioning and copying engines are in charge of deploying required resources at new cloud sites or making resource adjustments to existing sites.

III. Providing Geo-Elasticity Using GeoScale

In this section, we first outline three challenges and our propose approaches in designing geo-elasticity; then we explain our proactive provisioning algorithm in detail.

First, GeoScale needs to monitor and predict global workload distribution. To do so, GeoScale monitors application's incoming requests at each data center location by collecting request logs from server replicas. These request logs contain information such as a time-stamp, client IP address, requested URL, service time and response time seen by that request. GeoScale then determines the workload volume seen by each data center by mapping clients to their closet data centers. The distance between a client and a data center location is estimated by leveraging IP geolocation techniques. If a data center does not receive sufficient traffic (justifying deployment of an application replica), it is removed from the candidate data center set. GeoScale iteratively refines the set until all data centers in the set have at least a threshold amount of traffic mapped onto them. Based on an application's workload distribution, GeoScale determines the future (peak) workload that will be seen by each cloud site by using forecasting techniques such as ARIMA.

Second, GeoScale has to figure out *where* and *how many* resources to provision corresponding to the geo-dynamic workload. To do so, GeoScale provisions server capacity for each cloud site with queueing-based proactive as well as agile reactive provisioning methods. Proactive provisioning operates at longer time scales and handles long-term workload trends observed at these time scales; while reactive provisioning makes agile changes to current provisioned server capacities. Essentially, GeoScale varies the number of cloud locations based on workload spatial variations predicted; and determine the server capacity (i.e., number of servers) required at each location using our proactive algorithm in Section III-A.

Third, GeoScale is expected to execute provisioning promptly and configure new replicas to work with the entire application. To do so, GeoScale uses public Cloud APIs to start new servers from local VM images based on above geo-elastic provisioning decisions. In scenarios where provisioning processes involve cross-data center copying, GeoScale employs three optimizations that reduce provisioning latencies: (1) pre-copying large VM images, (2) selecting data center pairs based on available bandwidth, and (3) using a faster storing service [3].

A. Proactive Geo-Elastic Provisioning

Our proactive provisioning algorithm is responsible for determining which cloud sites to host application replicas and how many server resources are needed.

Proactive provisioning uses workload predictions to drive the provisioning algorithm. When workload is observed near a new cloud location, our algorithm makes decisions to start up one or more servers at this new location. Similarly diminishing workload near an existing cloud site may cause servers to be shut down, with the residual traffic from that region redirected to another close cloud location hosting replicas. In this way, proactive provisioning provides geo-elasticity by using observed changes in the spatial distribution of the workload to vary the number of *cloud locations* that house replicas of the application. This is in addition to handling changes in the temporal distribution in load at existing locations, which is handled by scaling the number of servers at those sites up or down.

Next, GeoScale employs a model-driven approach to determine the server capacity (i.e., number of servers) required at each location. Let λ_j^p denote the peak workload that will be seen by this location j as per the workload forecasting engine. We employ a $G/G/1$ queueing model of an individual server to determine the maximum request rate λ_j^c that can be serviced by a *single cloud server* without violating the application's SLA. We use the Kingman's theorem [4] for G/G/1 queue under heavy traffic that states waiting time W is an exponential distribution with mean $E[W] = \frac{\sigma_a^2+\sigma_b^2}{2(\frac{1}{\lambda}-\bar{x})}$; where σ_a^2 and σ_b^2 denote the variance in the requests inter-arrival time and service time, and λ and $\bar{x}$ represent the request arrival rate and mean service time

seen by this queueing system. Suppose SLA y is defined as the 95^{th} percentile of server response time, we derive the upper bound on the maximum rate λ_j^c under heavy traffic as shown in Equation 1.

$$\lambda_j^c < \left[\bar{x}_j + \frac{3(\sigma_{ja}{}^2 + \sigma_{jb}{}^2)}{2\,(y - \bar{x}_j)}\right]^{-1} \quad (1)$$

GeoScale uses empirical measurements from workload monitoring component to drive the above queueing models: estimating inter-arrival times variance $\sigma_{ja}{}^2$ and service times variance $\sigma_{jb}{}^2$. The request service times $\bar{x}_j$ at location j can be computed from server logs or measured by profiling the application on the server; if location j is a new location with no previous history, the observed service time from an existing nearby cloud location can be used as the initial estimate. The SLA y is specified by the application provider as the upper bound of response time that should not be violated, in our case 95^{th} percentile of server response time. Since all terms of Equation 1 are either known or empirically measured, we successfully obtain the maximum request rate λ_j^c that can be handled by a single server.

Last, we calculate the number of servers S_j required at location j to handle a peak request rate of λ_j^p with $S_j = \left\lceil \frac{\lambda_j^p}{\lambda_j^c} \right\rceil$. If S_j is greater than the current number of servers $\hat{S}_j$ provisioned at location j, then the provisioning algorithm needs to scale up capacity by allocating $(S_j - \hat{S}_j)$ additional servers. If $S_j < \hat{S}_j$, then capacity is scaled down by deallocating $(\hat{S}_j - S_j)$ servers. If $\hat{S}_j = 0$, this is a newly chosen cloud location for the application, and S_j new servers need to be started up at this location j. At the end of proactive provisioning, $\hat{S}_j$ is set to S_j.

IV. Experimental Evaluation

We evaluate the efficacy of GeoScale's queueing-based capacity model and proactive provisioning approach. Our evaluation demonstrate the accuracy of models for a set of cloud locations, and benefits of using geo-elasticity in improving end-users response time.

A. Experiment Setup

We run our system GeoScale as a middleware on top of Amazon's distributed cloud platforms. GeoScale uses APIs provided by Amazon EC2 and can manage server resources in eight data center locations. We use a java implementation of TPC-W as a representative multi-tier web application for benchmarking both model validation and GeoScale's ability to handle geo-dynamic client workload. Here, TPC-W benchmark emulates an online bookstore and employs a two-tier architecture, a web server tier based on Apache Tomcat and a database tier based on MySQL. We run each TPC-W replica inside small server instances and employ eventual consistency model across replicas. We run TPC-W client workload generators on PlanetLab nodes to simulate a geographically diverse workload. For each experiment run, we monitor and measure system-level statistics such as request rates required by capacity model as well as response time perceived by the clients.

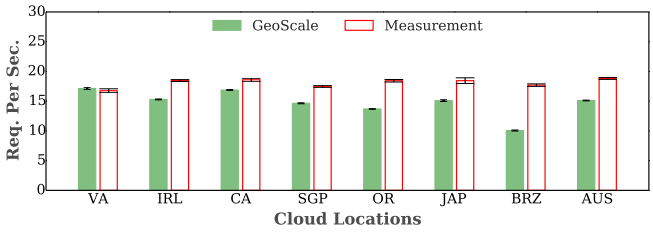

Figure 2: Comparisons of GeoScale's capacity modeling with empirical measurement for small server instances.

B. Capacity Model Analysis

We empirically evaluate our queueing-based model by comparing measured server capacity to model prediction. For each run, we host TPC-W on a small server instance in one of the cloud locations and start clients on a nearby PlanetLab node. We warm-up the application for five minutes and then steadily increase the workload until GeoScale detects SLA violations (i.e. 95^{th} percentile response time is greater than 1 second). We record the corresponding request rate as the server capacity. Concurrently we collect required system-level statistics and calculate server capacity based on queueing model described in Section III-A. We repeat above setup for 5 times for all eight cloud locations and plot the average results and 95^{th} confidence interval.

Figure 2 compares GeoScale's predicted server capacity with the empirically measured ones for each cloud location. The measured server capacity varies across cloud locations, with up to a 12% difference in capacity across locations for the same type of server. Because data centers were built in different years, we attribute these difference to variations in the underlying server hardware deployed at different locations. Figure 2 also emphasizes the need for a separate location-specific model to fully capture the hardware idiosyncrasies across location for the same type of server.

C. Geo-elastic Proactive Provisioning Benefits

We compare GeoScale's proactive provisioning approach with two variants of local elasticity where the choice of cloud locations is made manually: (i) single-site elasticity (SSE) and (ii) multi-site elasticity (MSE). SSE is a centralized approach that hosts all application replicas at a *single* cloud location, while MSE houses replicas at a predetermined *static* set of locations. We assume all three provisioning approaches employ the queuing model described in Section III-A to handle temporal workload fluctuations.

We run TPC-W clients on PlanetLab nodes from three locations: Pennsylvania and California in the USA and

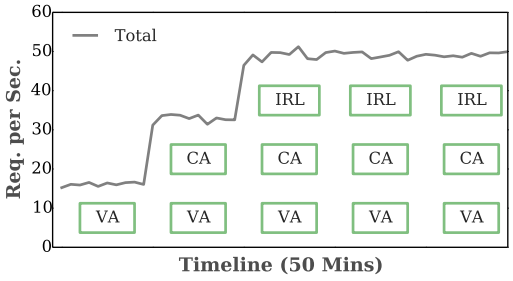

Figure 3: Illustration of client workload dynamics and proactive provisioning in three data centers.

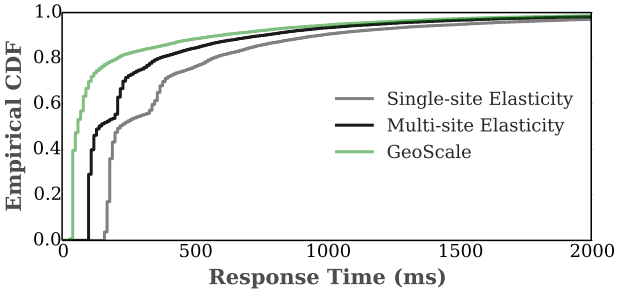

Figure 4: ECDF of client-perceived response time.

Germany in Europe. Pennsylvania clients starts sending requests followed by California clients at $t = 10$ minutes and Germany clients at t=20 minute. The aggregate application workload is illustrated in Figure 3. All three proactive approaches are able to scale up server capacity in response to the workload increase. The main difference is *where* the servers are provisioned. The SSE technique centralizes all replicas at Amazon's Virginia (VA) cloud location and scales server capacity from 1 server to 3 servers at this site to handle the workload increase. The MSE approach is configured with replicas at Amazon's VA and California (CA) data centers and it provisions one server in CA and two servers in VA to handle workload increase. GeoScale's proactive elasticity technique allocates one server in Amazon's VA data center to handle the Pennsylvania clients, followed by another server in CA location to handle California traffic, and a third server in Amazon's Ireland (IRL) data center to handle the traffic from Germany(Figure 3).

Figure 4 shows client response times CDF for all three provisioning approaches. SSE has the highest response times since it uses a single cloud location to serve global traffic, causing distant clients to see worse response times. MSE approach uses a couple fixed locations to host the application and is able to direct clients to the closer of the two locations, yielding better response times than SSE. GeoScale yields the best response times since it is able to provision servers that are closest to the clients. The 95^{th} percentile response time provided by GeoScale is around 1060 ms, a 31.17% improvement over SSE and a 13.11% improvement when compared to MSE.

V. Related Work

Prior work [5]–[7] studied placing application services closer to the end-users to reduce network latency, in the context of one-time placement or static contents. Our focus is on dynamic capacity provisioning to handle both temporal and spatial workload variations by building on top of VM-based elasticity techniques [8]–[10]. At the core of our work is queueing-based models that estimate required server capacities in different cloud locations, derived from previous single data center models [11]–[14].

VI. Conclusion and Future Work

In this paper, we presented challenges and approaches for implementing geo-elasticity in distributed clouds. Our system GeoScale employs a queueing model-driven proactive provisioning technique. Our initial experiments using Amazon's distributed clouds showed average 88% model accuracy and 31% user-perceived improvement. As part of future work, we plan to conduct detailed evaluations of GeoScale and integrate supports for vertical scaling of heterogeneous resources.

References

[1] T. Kraska *et al.*, "Mdcc: Multi-data center consistency," in *EuroSys*, 2013.

[2] M. Patiño Martinez *et al.*, "Middle-r: Consistent database replication at the middleware level," *ACM TOCS*, 2005.

[3] T. Guo *et al.*, "Geoscale: Providing geo-elasticity in distributed clouds," School of Computer Science, Univ. of Massachusetts at Amherst, Tech. Rep. UM-CS-2015-009, April 2015.

[4] J. F. C. Kingman, "The single server queue in heavy traffic," *Math. Proc. Cambridge Philos. Soc.*, 1961.

[5] L. Qiu *et al.*, "On the placement of web server replicas," in *INFOCOM*. IEEE, 2001.

[6] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, 2009.

[7] "Content Delivery Network," http://www.akamai.com/html/resources/content-distribution-network.html.

[8] C. Clark *et al.*, "Live migration of virtual machines," in *NSDI*, 2005.

[9] T. Knauth *et al.*, "Scaling non-elastic applications using virtual machines," in *CLOUD*, 2011.

[10] H. A. Lagar-Cavilla *et al.*, "Snowflock: Rapid virtual machine cloning for cloud computing," in *EuroSys*, 2009.

[11] D. Gmach *et al.*, "Workload analysis and demand prediction of enterprise data center applications," in *IISWC*, 2007.

[12] A. Gandhi *et al.*, "Adaptive, model-driven autoscaling for cloud applications," in *ICAC*, 2014.

[13] S. J. Malkowski *et al.*, "Automated control for elastic n-tier workloads based on empirical modeling," in *ICAC*, 2011.

[14] T. Guo *et al.*, "Model-driven geo-elasticity in database clouds," in *ICAC*, 2015.