Modeling the Impact of Workload on Cloud Resource Scaling

Anshul Gandhi[†*], Parijat Dube[†], Alexei Karve[†], Andrzej Kochut[†], Li Zhang[†]
[†] IBM T. J. Watson Research Center, Yorktown Heights, NY, 10598, {gandhian,pdube,karve,akochut,zhangli}@us.ibm.com
[*] Stony Brook University, Stony Brook, NY, 11790, anshul@cs.stonybrook.edu

***Abstract*—Cloud computing offers the flexibility to dynamically size the infrastructure in response to changes in workload demand. While both horizontal and vertical scaling of infrastructure is supported by major cloud providers, these scaling options differ significantly in terms of their cost, provisioning time, and their impact on workload performance. Importantly, the efficacy of horizontal and vertical scaling critically depends on the workload characteristics, such as the workload's parallelizability and its core scalability. In today's cloud systems, the scaling decision is left to the users, requiring them to fully understand the tradeoffs associated with the different scaling options. In this paper, we present our solution for optimizing the resource scaling of cloud deployments via implementation in OpenStack. The key component of our solution is the modeling engine that characterizes the workload and then quantitatively evaluates different scaling options for that workload. Our modeling engine leverages Amdahl's Law to model service time scaling in scale-up environments and queueing-theoretic concepts to model performance scaling in scale-out environments. We further employ Kalman filtering to account for inaccuracies in the model-based methodology, and to dynamically track changes in the workload and cloud environment.**

I. Introduction

Cloud computing provides users the opportunity to quickly deploy their applications/workloads in an elastic setting using a "pay-as-you-go" pricing model. Cloud-deployed workloads can be easily scaled in response to changing workload demand by leveraging the cloud framework (see, for example [1], [2]). Cloud workloads are scaled up when the workload demand increases to handle the additional traffic without violating performance Service Level Agreements (SLAs), and can be scaled down when the workload demand decreases to save on rental costs.

An application or workload can be horizontally scaled ("scale-out") by adding more virtual machines (VMs) to the existing deployment, and can be vertically scaled ("scale-up") by adding more resources (such as vCPUs, memory, etc) to the existing VMs. The choice of scaling can have a significant impact on the cost and performance of the workload. For example, we show in Section V that scale-up can be 3 times more cost-effective than scale-out when the workload demand is low, whereas scale-out can be 2 times more cost-effective when the workload demand is high. However, the results also depend on the performance SLA and the cost function[1]. Thus, it is not at all obvious as to how the workload should be scaled in response to varying traffic conditions.

[1]We acknowledge that the per-VM prices of various cloud providers depend on external factors such as profit margins and market fluctuations. As such, we will often use the amount of resources employed by the workload or application as a proxy for the "cost", and only use cloud providers' listed prices for specific use cases.

Importantly, the choice of scaling also depends on the workload characteristics. *Parallel workloads* that can be parallelized on multiple cores to reduce their service time can significantly benefit from scale-up. *Sequential workloads*, on the other hand, do not benefit from multiple cores in terms of service time. However, they do benefit from increased concurrency due to scale-up. Given these differences, *we assert that the optimal scaling solution should account for the impact of workload characteristics* on the resulting performance. One approach to scaling is to try all possible scaling options and then pick the most cost-effective option that meets the performance SLA. This is clearly an ineffective approach, which is made even more complicated by the fact that a scaling decision can involve scale-out and scale-up *simultaneously*, thereby significantly increasing the space of possible scaling options.

There have been several empirical studies [3], [4], [5] aimed at understanding the scalability of workloads on multi-core, multi-thread systems in scale-up architectures. Most of these studies are targeted at characterizing workload performance in terms of system architecture (CPU cores, SMT level, cache size, memory size and type) and in identifying resource contentions in the software and/or hardware stack. While some models linking workload performance to specific architecture features have been proposed [6], [7], [8], to the best of our knowledge, no attempts have been made to leverage workload characterization for optimizing horizontal and/or vertical resource scaling in clouds.

In this paper, we propose a model-based approach to assess the impact of workload on cloud resource scaling. The key component of our solution is the modeling engine that characterizes the workload and its horizontal and vertical scalability. Our modeling engine leverages Amdahl's Law to model service time scaling in scale-up environments and queueing-theoretic concepts to model performance scaling in scale-out environments. Using our model, we dynamically determine the most cost-effective scaling option, including combinations of scale-out and scale-up, that will result in the required performance for any given workload.

The main contributions of this paper are:

- We experimentally investigate the impact of horizontal and vertical scaling on cost, performance, and provisioning times for different cloud-deployed applications (Section III).
- We develop a performance model using Amdahl's Law and queueing-theoretic principles to predict the tradeoff of various scaling options (Section IV). We also employ a Kalman filtering-based approach to dynamically infer the (possibly changing) system parameters required by our model with minimal benchmarking efforts (Section IV). The Kalman filter also provides robustness against inaccuracies that are inherent in model-driven approaches.

- We validate our solution via implementation on OpenStack, and demonstrate the benefits of our approach by determining the optimal scaling of popular multi-tier benchmark workloads (Section V).

II. Experimental Setup

We use OpenStack [9] as the underlying scalable cloud system. The VMs for the applications are created on an OpenStack managed private cloud deployment on SoftLayer [2] physical machines. We employ multiple hypervisors with 8 CPU cores and 8 GB of memory each. Scale-up and scale-out is executed via the OpenStack API. We use Chef [10] to automate the installation of software on VMs during boot. At a high-level, applications are deployed as a collection of VMs in OpenStack. We experiment with two different application workloads: (i) RUBiS [11], which is a sequential workload, and (ii) LINPACK [12], which is a parallel workload. We discuss the specifics of these workloads and their deployment in Section II-A. We monitor the resource consumption and application performance using our Monitoring agent, described in Section II-B. The monitored statistics are then forwarded to the Modeling engine, described in Section II-C, which performs the workload characterization and analyzes the different scaling options.

A. Workloads

1) RUBiS: We use the open source multi-tier application, RUBiS [11], as a representative sequential workload for our experiments. RUBiS is an auction site prototype modeled after eBay.com supporting 26 different classes of web requests such as bid, browse, buy, etc. Our implementation of RUBiS employs Apache as the front-end web server, Tomcat as the Java servlets container, and MySQL as the back-end database. In our experiments, we focus on scaling the Tomcat tier. We employ RUBiS's benchmarking tool, which emulates user behavior by defining sessions consisting of a sequence of requests, to generate load for our experiments. The think time between requests is exponentially distributed with a mean of 1 second. We fix the number of clients for each experiment and vary the load by dynamically changing the composition of the request classes mix.

The Apache and MySQL tiers are each hosted on a 4 vCPU VM. The Tomcat application tier is hosted on multiple VMs. We use host aggregates and availability zones (which are essentially logical cloud partitions) offered by the OpenStack nova scheduler to place the Apache and MySQL VMs on one hypervisor and Tomcat VMs on a different set of hypervisors.

We set the SLA in terms of mean response time. We focus on the mean response time of browse requests since customers often base their web experience based on how long it takes to browse through online catalogues. We want the response time SLA for the browse requests to be less than 35ms, on average. The secondary goal is to minimize the total cost of Tomcat VMs employed by the application. We use the total number of vCPUs employed by the application tier as a proxy for cost, unless stated otherwise. Thus, the overall objective is to minimize the resource cost while ensuring that the 35ms SLA for the browse requests is not violated. In our experiments, we find that the application performance is most affected by the processing power (number of VMs and number of cores per VM), and is much less affected by the allocated memory and storage space. We thus focus on scaling the processing power of the application tier.

2) LINPACK: We use Intel LINPACK [12] as a representative parallel workload for our experiments. LINPACK executes a collection of linear algebra operations on matrices of a given input size. The workload is inherently parallelizable, with a small sequential portion during the initialization phase. We run LINPACK in a transactional manner by using httperf [13] as the workload generator and Apache as the front-end load balancer that distributes requests to back-end application servers running LINPACK.

The httperf application and the Apache load balancer are each hosted on a 4 vCPU VM. The LINPACK application tier is hosted on multiple VMs. We set the mean response time SLA for LINPACK requests as 5 seconds. The overall objective is to minimize the resource cost of the LINPACK application tier while ensuring that the 5s SLA is not violated. Since LINPACK is CPU-bound, we focus on scaling the processing power of the LINPACK VMs.

B. Monitoring agent

While most of the monitoring information required for our model-driven approach can be obtained directly from the physical infrastructure, some of the more critical information, such as per-tier service times, requires invasive benchmarking which is not possible for a cloud service provider. For estimating such unobservable parameters, we employ Kalman filtering (see Section IV).

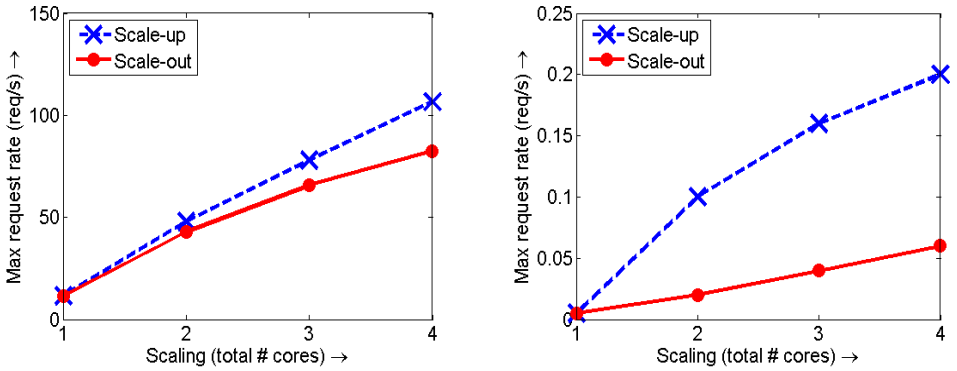
We use virt-top (part of the libvirt [14] package) to collect VM CPU utilization statistics from each hypervisor periodically. For the application-level metrics, we periodically monitor the request URLs directed at the application deployment to compute the request rate and response time. Note that the user can choose to provide these metrics to us directly (for example, using a REST call). The monitoring interval is set to 10s.

C. Modeling engine

The modeling engine is responsible for workload characterization and analysis of the various scaling options. The modeling engine is invoked whenever a new entry is recorded by the monitoring engine (once every 10s). The modeling engine consumes this information and infers the important system parameters via the Kalman filtering approach (Section IV). The underlying queueing model, along with the inferred parameters, allow the modeling engine to determine the optimal scaling action, if any is needed.

III. Performance-Cost Tradeoffs

In this section, we investigate the effects of scale-up and scale-out on performance and cost. For performance, we look at the maximum load, or request rate, that the system can handle without violating the response time SLA (35ms for RUBiS and 5s for LINPACK). For cost, we look at hourly rental prices based on Amazon EC2's General Purpose On-Demand Instances [15], Rackspace's Performance 1 Flavor Managed Cloud Service [16], and Softlayer's CloudLayer Instances [17].



(a) Sequential workload RUBiS [11] (b) Parallel workload LINPACK [12]

Fig. 1: Performance of scale-up and scale-out in terms of maximum allowable request rate for a fixed SLA.

We also consider the provisioning time, which is the time taken to execute the scaling action.

A. Performance

Figure 1 shows our experimental results for the maximum request rate that can be handled by the application (without violating the SLA) as a function of the scaling. The scaling is in units of total number of cores employed by the application tier. In particular, scaling refers to the total number of vCPUs on the single VM in case of scale-up and the number of 1vCPU VMs in case of scale-out.

In Figure 1(a), we see that performance scales almost linearly under RUBiS for both scale-out and scale-up. Since RUBiS is a sequential workload, adding more cores to the VM or adding more VMs has the same effect: increased computing power to handle greater concurrency. There are, of course, difference in the two different types of scaling. Scale-up introduces more contention for OS and locks, whereas scale-out requires more work at the load balancer.

In Figure 1(b) for LINPACK, we see that performance increases significantly under scale-up initially, and then starts to plateau. We claim that this concave downwards shape is because of diminishing gains obtained from scale-up. Under scale-out, performance scales linearly. However, for LINPACK, we see that scale-up results in much better performance than scale-out. This is because LINPACK is parallelizable, and thus, scale-up *decreases* the service time of LINPACK. This allows for more aggressive sharing of resources *at* the VM. By contrast, under scale-out, the service time of a single LINPACK request does not change. Thus, to get good response times, a VM should only host one request at any time.

Observe that the frequency of a single vCPU is not changed in our setting as we are not exploiting dynamic frequency scaling [18]. Dynamically changing frequency of vCPUs as an alternate to increasing number of vCPUs (with fixed, static frequency) is also been employed in existing solutions (see e.g., VMware vCenter [19]). This will add another dimension in the scaling decision problem and may result in different scaling behavior for sequential workloads like RUBiS. However, CPU frequency scaling is unappealing due to its limited range [20] and diminishing returns, since higher clock frequency causes more energy dissipation [18].

The observed differences in performance of scale-up and scale-out for the sequential and parallel workloads in Figure 1 motivates us to develop a model to better understand the impact of workload on scaling. We explain our modeling and analysis in Section IV.

B. Cost

Figure 2 shows the per-instance hourly rental costs for different cloud providers. We obtain the cost of scale-out by linearly extrapolating the cost of a single VM. As such, we do not take into account any discounts that might be possible because of bulk orders. We see that the cost function is exactly the same (linear) for scale-up and scale-out under Amazon EC2's pricing model. We observed the same linear cost function under Microsoft Azure's pricing scheme as well [21] (not shown). In such cases, the decision between scale-up and scale-out depends on the performance impact. For Rackspace [16] and Softlayer [17], we find that the scale-up cost is lower than the scale-out cost for the same total number of cores purchased. Note that this analysis does not take other resources such as memory and storage into account. In choosing the above data, we tried to ensure that the memory and storage specifications were constant across different service providers for each CPU scaling size.

C. Provisioning time

In terms of the provisioning time, it takes about 30-40 seconds for scale-out in our OpenStack environment (via the `boot`

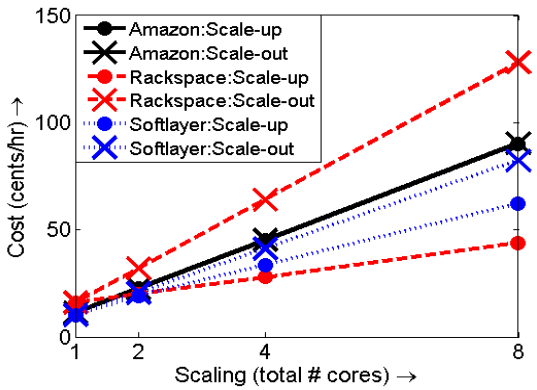
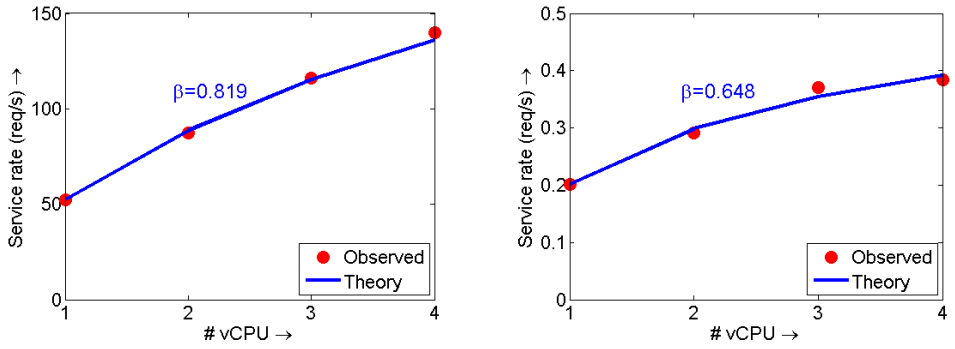


Fig. 2: Cost of scale-up and scale-out in terms of cents/hr.



(a) RUBiS (b) LINPACK

Fig. 3: Modeling service rate scale-up via Amdahl's Law. β represents the scale-up efficiency of the workload. Average modeling error is 1.3% in case of (a) the sequential workload RUBiS and 2.3% in case of (b) the parallel workload LINPACK.

command in OpenStack) with pre-configured application tier images. However, if vanilla images are used and application software is installed after boot-up, the provisioning time can easily extend to 4-10 minutes. For scale-up, the provisioning time is about 20 seconds. During scale-up (via the `resize` command in OpenStack), the instance is rebooted with the new resource configuration. While this results in application down-time during scale-up, we do not have to reinstall application software.

IV. Modeling

The modeling engine lies at the heart of our solution. Our modeling goal is to understand how scale-out and scale-up affect response time.

A. Modeling scale-out

We use a queueing-network model [22] to *approximate* our multi-tier cloud workloads. Consider the RUBiS workload described in Section II-A1. For RUBiS, we have a single front-end VM and a single database VM, but possibly many application tier VMs. We assume perfect load balancing among the homogeneous application tier VMs, thus allowing us to approximate the application tier using a single representative M/G/1/PS [22] queue. If the total workload request rate is λ, then the request rate at this single M/G/1/PS queue is $\frac{\lambda}{k}$, where k is the number of application tier VMs. Under this model, the mean response time, T, is [22]:

$$T = \left(\frac{1}{S_{fe}} - \lambda\right)^{-1} + \left(\frac{1}{S_{app}} - \frac{\lambda}{k}\right)^{-1} + \left(\frac{1}{S_{db}} - \lambda\right)^{-1} \quad (1)$$

where S_{fe}, S_{app}, and S_{db}, are the service times[2] at the front-end, application, and database tiers, respectively. Equation (1) tells us how scale-out affects response time.

B. Modeling scale-up

In order to understand the effect of scale-up on response time, we first have to analyze how service time at the application tier, S_{app}, is affected by scale-up. Amdahl's Law [6] suggests that scale-up should decrease a fraction, β, of S_{app} multiplicatively, assuming the workload is parallelizable, for example LINPACK (see Section II-A2). This β fraction refers to the portion of S_{app} that is parallelizable. In particular, if $S_{app}(x)$ denotes the application tier service time when the application tier VMs have x vCPUs each, then:

$$S_{app}(x) = S_{app}(1) \cdot \left(\frac{\beta}{x} + (1 - \beta)\right) \quad (2)$$

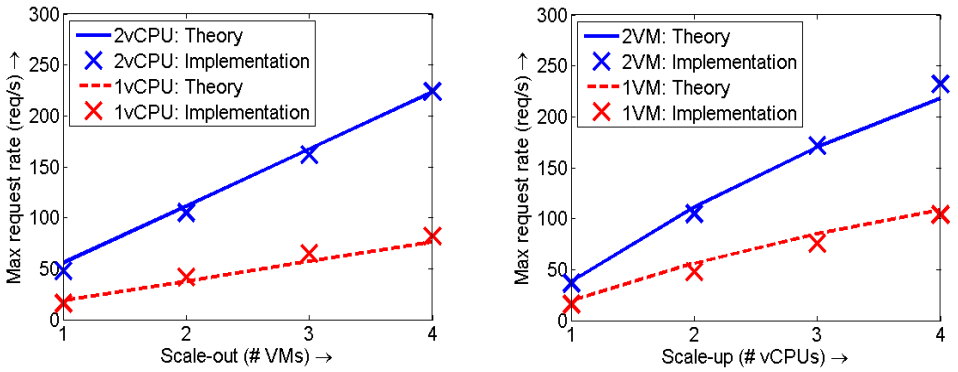
Interestingly, even for sequential workloads that cannot be parallelized, such as RUBiS, we can use a similar approach. Such workloads benefit from increased concurrency due to multiple cores. In this case, while the service time remains the same, the service rate (or peak throughput) of the VM increases. We expect the service rate to be proportional to the number of vCPUs, x. Since service time can be viewed as the inverse of service rate, we can use Equation (2) to also model the increase in service rate for sequential workloads such as RUBiS. As such, β represents the scale-up efficiency of a workload. Our results in Section IV-E validate our service time/rate modeling for sequential and parallel workloads.

C. Kalman filtering and robustness

Cloud providers cannot access the user application directly to compute system parameters such as service times. We thus use a Kalman filtering technique to infer these unobservable parameters. Kalman filtering works by leveraging observable monitoring information, such as end-to-end response time and VM utilization (see Section II-B), and using model-based equations, such as Equation (1), to estimate unobservable parameters. Importantly, by employing the Kalman filter to leverage the *actual* monitored values, we minimize our dependence on the approximate queueing model of the system. Another critical advantage of employing a Kalman filter in cloud performance modeling is the ability to quickly detect and react (by updating the system parameters) to changes in the workload, as evidenced by our previous work [23]. Due to lack of space, we omit details on Kalman filtering and refer the readers to [24], [23] for more information.

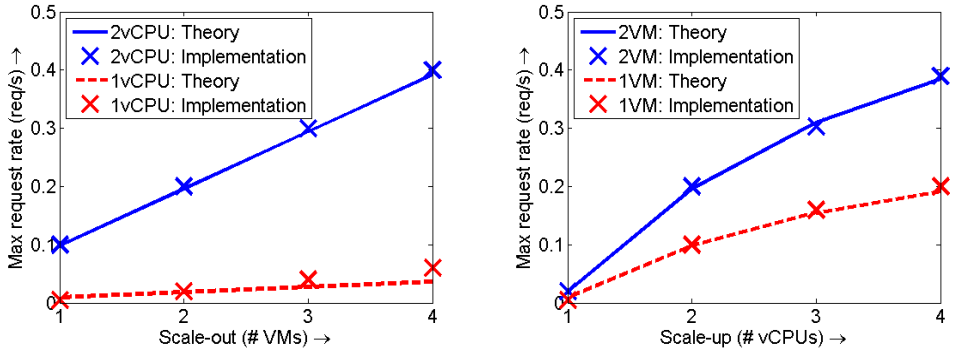
We employ Kalman filtering to estimate service times (and service rates). In particular, we run experiments with varying request rates for each scale-up setting (with 1 VM only) and process the collected monitored data (see Section II-B) via Kalman filtering to estimate per-tier service times. This

[2]Service time for a tier is defined as the total time taken to serve the workload request at that tier, assuming no resource contention. In other words, it is the minimum execution time at a tier.



(a) Scale-out (b) Scale-up

Fig. 4: Good agreement (9% error) between theoretical and implementation results for the sequential workload RUBiS.



(a) Scale-out (b) Scale-up

Fig. 5: Good agreement (7% error) between theoretical and implementation results for the parallel workload LINPACK.

profiling step can be avoided if we have prior knowledge of β and $S_{app}(1)$. Figure 3 shows our estimated service times (crosses) based on observed experimental results with scale-up for (a) the sequential workload RUBiS and (b) the parallel workload LINPACK. We use our OpenStack implementation setup described in Section II for the experiments. We then use regression to fit (shown as solid line) the estimated service times to Equation (2). This gives us $\beta = 0.819$ and $S_{app}(1) = 21ms$ for RUBiS with a modeling error of 1.3% (parallelism at the HTTP request level, allowing execution of multiple concurrent requests) and $\beta = 0.648$ and $S_{app}(1) = 4.96s$ for LINPACK with a modeling error of 2.3% (parallelism at the thread level, but with a small sequential initialization portion).

D. Combined model

Combining Equations (1) and (2), we get the final model for response time as a function of workload characteristics (β and $S_{app}(1)$) and scale-up (x) and scale-out (k). In particular, we replace S_{app} in Equation (1) with $S_{app}(x)$ from Equation (2) to derive T as a function of x (scale-up) and k (scale-out). We can use this $T(x,k)$ function to determine the maximum request rate that can be handled by a given scaling option without violating the SLA (see Section IV-E), which is the largest value of λ in Equation (1) such that (s.t.) $T(x,k) < SLA$. For a given λ, we can obtain feasible values of x and k such that $T_\lambda(x,k) < SLA$, and further, combine this function with a cost function, such as those in Figure 2, to determine the most cost-effective scaling option (see Section V).

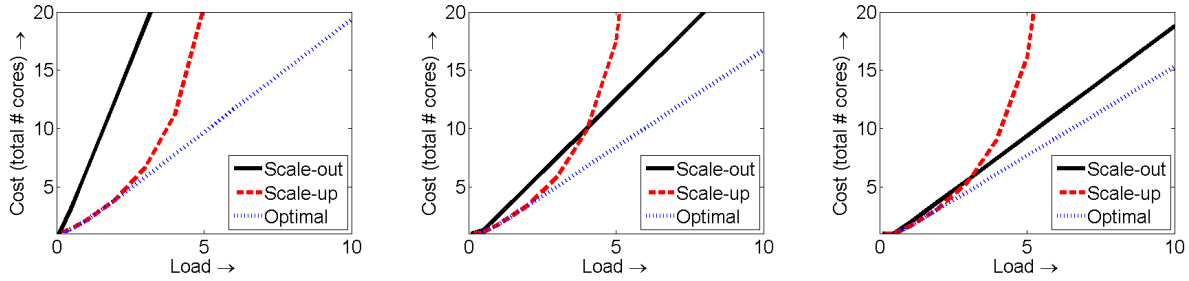
E. Model validation

Figures 4 and 5 show our theoretical and implementation results for (a) scale-out and (b) scale-up for RUBiS and LINPACK respectively. We use our OpenStack implementation setup described in Section II for these experiments. We see that our model can predict performance (in terms of maximum load that can be sustained without violating SLA) fairly well for both scale-out and scale-up, including combinations of the two. The average prediction error is about 9% for RUBiS and 7% for LINPACK. In the next section, we employ our model to predict the optimal scaling under various scenarios.

V. Analysis

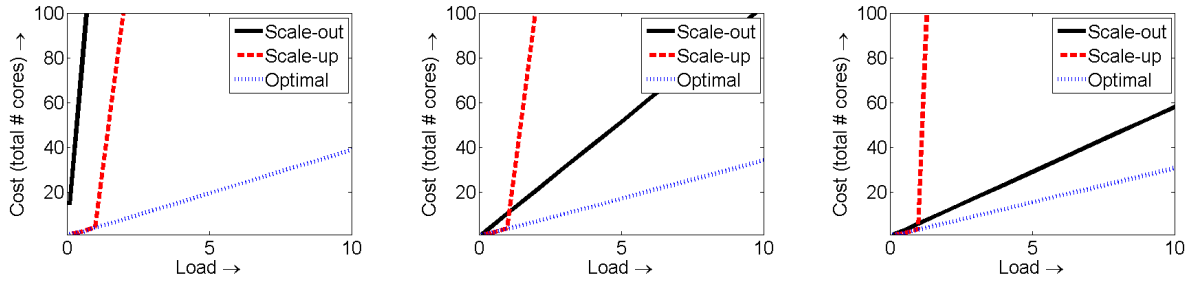
A. Comparison of different scaling options

We now employ our scaling model to examine the various scaling options for different workloads. Figure 6 shows our analytical results for RUBiS in terms of cost as a function of the load under scale-out, scale-up, and optimal, for different response time SLAs. A load of l is defined as the request rate (λ_l) that equals l times the peak throughput of a single 1-vCPU VM (λ_0). Here, optimal refers to the most cost-effective combination of scale-out and scale-up, as determined by our model, that satisfies the SLA. Mathematically, the optimal configuration for load l is a solution of the following optimization problem: $\min_{x,k} c(x,k)$, s.t. $T_{\lambda_l}(x,k) < SLA$, where, $\lambda_l = l * \lambda_0$, and $c(x,k)$ is the cost of renting k VMs with x vCPUs each from the cloud provider. For cost, we use the total number of cores employed as opposed to the more volatile metric of cents/hr.



(a) SLA = 25ms (b) SLA = 35ms (c) SLA = 45ms

Fig. 6: Analysis of various scaling options for RUBiS as a function of load for various response time SLAs.



(a) SLA = 5.0s (b) SLA = 5.5s (c) SLA = 6.0s

Fig. 7: Analysis of various scaling options for LINPACK as a function of load for various response time SLAs.

We observe that the combination of scale-out and scale-up (optimal) can provide significant cost savings over the individual scaling options. The cost savings increase with load (or request rate) and are higher when the response time SLA is more strict. Further, scale-up is more cost-effective than scale-out when the SLA is more strict and when the load is low. However, scale-out is more cost-effective than scale-up when the load is high.

For the case of 25ms SLA in Figure 6(a), scale-up is the optimal choice at low request rates, and provides significant cost-savings over scale-out (about 64% at a load of 2 for 1 5-vCPU VM vs 14 1-vCPU VMs). However, scale-up *cannot* achieve the required response time at a load of above 7 due to the nature of service time scaling explained in Section IV. For high loads, scale-out is superior to scale-up, but is still significantly more expensive than optimal (more than 3 times costlier than optimal at a load of 8 for 53 1-vCPU VMs vs 4 4-vCPU VMs). As the response time target becomes less strict in Figures 6(b) and 6(c), scale-out starts to approach optimal. The cost savings of optimal over scale-out for response time targets of 35ms and 45ms are about 29% (5 3-vCPU VMs vs 21 1-vCPU VMs) and 25% (6 2-vCPU VMs vs 16 1-vCPU VMs) respectively, at a load of 8.

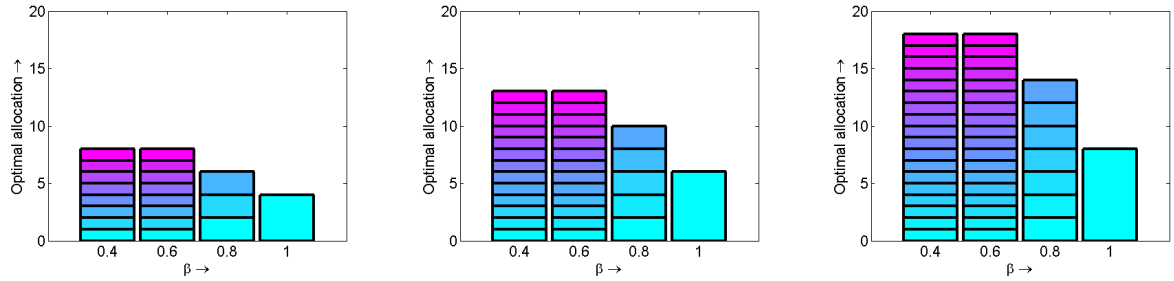
Figure 6 shows our analytical results for LINPACK in terms of cost as a function of the load under scale-out, scale-up, and optimal, for different response time SLAs. We again see that optimal provides significant cost savings over individual scaling options, and the savings increase with load and are greater when the SLA is more strict. For the case of 5s SLA in Figure 7(a), scale-up is more cost-effective than scale-out. This is to be expected based on our initial experimental results in Figure 1(b). However, for the case of 5.5s SLA and 6s SLA in Figures 7(b) and 7(c) respectively, we see that scale-out outperforms scale-up. This is because of the low β value (low scale-up efficiency) for LINPACK (see Figure 3(b)), which makes scale-up less desirable at high scale. This is to be expected since a high scale-up value implies more OS and lock contention at the VM as opposed to the "shared nothing" scale-out option.

B. Analysis of the optimal scaling option

We now further examine the optimal scaling in order to better understand the optimal resource allocation. Figure 8 shows the resource allocation under the optimal scaling policy for different loads as a function of β. Here, we assume the parameters of the RUBiS workload setup except for β (that is, SLA and $S_{app}(1)$). The number of rectangles in each vertical bar represents the number of VMs in that allocation, and the height of each rectangle represents the number of vCPUs. We restrict our analysis to homogeneous VM configurations only.

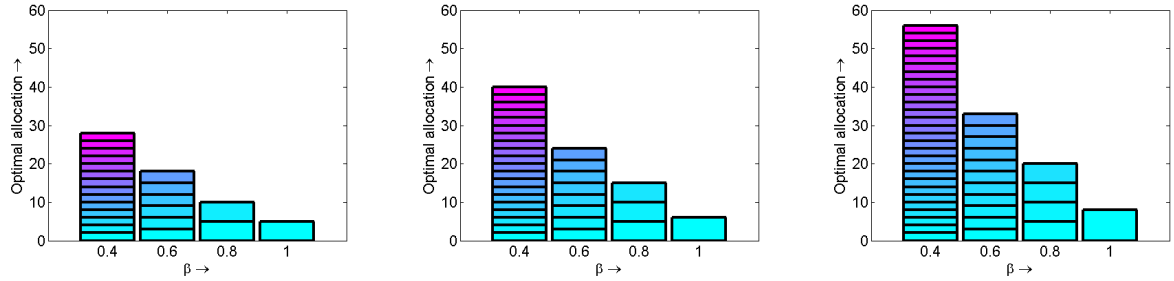
We see that the optimal scaling prefers larger VMs (scale-up) as β increases. This is to be expected since β represents the scale-up efficiency of the workload. Interestingly, we see that the optimal VM size (number of vCPUs per VM) for a given β does not change with load, except for the case of $\beta = 1$. Thus, the optimal allocation can be viewed as scaling out the optimal-sized VM based on the load. Importantly, we find that the cost (total number of vCPUs) of optimal scaling decreases with β. Thus, it is less expensive to scale workloads that have high scale-up efficiency.

Figure 9 shows the resource allocation under the optimal scaling policy for different loads as a function of β. Here, we assume the parameters of the LINPACK workload setup except for β. We again see that the optimal scaling prefers larger VMs as β increases, and the optimal VM size for a given β appears to be insensitive to load. We also see that the cost of optimal scaling decreases significantly with β.



(a) Load = 3 (b) Load = 5 (c) Load = 7

Fig. 8: Optimal allocation as a function of β for various loads assuming the RUBiS workload setup. The number of rectangles in each vertical bar represents the number of VMs and the height of each rectangle represents the number of vCPUs.



(a) Load = 3 (b) Load = 5 (c) Load = 7

Fig. 9: Optimal allocation as a function of β for various loads assuming the LINPACK workload setup. The number of rectangles in each vertical bar represents the number of VMs and the height of each rectangle represents the number of vCPUs.

VI. Related Work

There has been a lot of work on pure scale-out, or horizontal scaling [25], [26], [27], and pure scale-up, or vertical scaling [28], [29], [30], of applications. There has been much less work on combined scale-out and scale-up. A comparison of scale-out and scale-up architectures is presented in [31], [32], where the need for an automated approach for comparison among the two scaling options is advocated. The availability of different scaling options in the cloud makes the choice of scaling difficult, causing practitioners to often make incorrect choices [33]. The advantages and challenges of combined scale-out and scale-up were empirically analyzed in [34], [35], [36], [37]. An experimental evaluation of scale-out vs scale-up architectures for commercial workloads was carried out in [38], [39], [40] where it was concluded that scale-out offers better cost-performance tradeoff compared to scale-up. However, some recent studies [30], [41] argue that scale-up can achieve comparable, and sometimes, even better performance than scale-out for certain workloads. All of the above cited works focus solely on experimental research for evaluating the tradeoffs between scale-out and scale-up. To the best of our knowledge, our work is the first to propose and validate an analytical model for evaluating different scaling options.

VII. Conclusion and Future Work

In this paper, we present an analytical model to understand the impact of workload characteristics on the efficacy of scale-out and scale-up in the cloud. We examine the core scalability of sequential and parallel workloads, and model its effects on performance scaling. Our model provides great accuracy when compared with implementation results, and is very useful in determining the optimal scaling for any given workload as a function of workload demand and required SLA.

Our results indicate that combining scale-up and scale-out can provide significant cost savings over pure scale-up or pure scale-out. Further, we find that scale-up is typically superior to scale-out when the SLA is strict, whereas scale-out is typically superior to scale-up when the load is high. Importantly, we find that the relative ordering of the different scaling options depends critically on the workload characteristics. For workloads that have high scale-up efficiency, vertical scaling is near-optimal for all SLAs and loads. Further, the total amount of resources required by a workload for meeting its SLA decreases with an increase in the scale-up efficiency.

In our experiments with RUBiS, there are multiple request types concurrently running on the system. For simplicity, we only focus on the performance of a single request type (browse), and restrict our application performance model in Section IV to a single request type system. However, it is possible to extend our performance modeling to workloads with multiple request types, as in our previous work [23].

Our performance metric in this paper is mean response time. We can extend our approach to capture other metrics as well, such as higher percentiles of response time, or combinations of metrics such as response time and latency. However, we do require the performance metric(s) to be expressed in terms of system variables, as in Equation (1). An alternative approach would be to model the system as a black-box, and infer the relationship between performance and observable system parameters. Such a black-box modeling approach, however, does have its limitations. For example, a 100% cpu usage is not surprising for a batch workload, such as Hadoop, but is alarming, and indicates the need for scaling in the case of transactional workloads, such as RUBiS.

For the workloads in this paper, we only focused on scaling compute resources. In the future, we will consider workloads,

such as Hadoop, that require the scaling of multiple resources including compute, I/O, and network.

Our modeling efforts in this paper represent the first step towards the development of an automated optimal scaling solution for cloud applications. As part of future work, we will leverage our model to build an online scaling service for cloud applications that will execute the most cost-effective scale-out and scale-up actions in response to varying workload demand while taking the workload characteristics into account.

REFERENCES

[1] Amazon Inc., "Amazon Auto Scaling," http://aws.amazon.com/autoscaling.

[2] SoftLayer Technologies, Inc., http://www.softlayer.com.

[3] J. H. Tseng, H. Yu, S. Nagar, N. Dubey, H. Franke, P. Pattnaik, H. Inoue, and T. Nakatani, "Performance studies of commercial workloads on a multi-core system," in *Proceedings of the 2007 IEEE International Symposium on Workload Characterization*, Boston, MA, USA, 2007, pp. 57–65.

[4] H. Inoue and T. Nakatani, "Performance of Multi-Process and Multi-Thread Processing on Multi-core SMT Processors," in *Proceedings of the 2010 IEEE International Symposium on Workload Characterization*, Atlanta, GA, USA, 2010, pp. 209–218.

[5] X. Guerin, W. Tan, Y. Liu, S. Seelam, and P. Dube, "Evaluation of Multi-core Scalability Bottlenecks in Enterprise Java Workloads," in *Proceedings of the 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Arlington, VA, USA, 2012, pp. 308–317.

[6] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[7] J. E. Moreira, M. M. Michael, D. Da Silva, D. Shiloach, P. Dube, and L. Zhang, "Scalability of the nutch search engine," in *Proceedings of the 21st Annual International Conference on Supercomputing*, Seattle, WA, USA, 2007, pp. 3–12.

[8] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, apr 2009.

[9] Openstack.org, "OpenStack Open Source Cloud Computing Software," http://www.openstack.org.

[10] Opscode Inc., "Chef," http://www.opscode.com/chef.

[11] "RUBiS: Rice University Bidding System," http://rubis.ow2.org.

[12] Intel Corp., "Intel Math Kernel Library - LINPACK 11.1 Update 2," https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download.

[13] D. Mosberger and T. Jin, "httperf - A Tool for Measuring Web Server Performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 31–37, dec 1998.

[14] "libvirt virtualization API," http://libvirt.org.

[15] Amazon Web Services, Inc., "Amazon EC2 Pricing," http://aws.amazon.com/ec2/pricing.

[16] Rackspace, US Inc., "Cloud Servers Pricing - Rackspace Hosting," http://www.rackspace.com/cloud/servers/pricing.

[17] SoftLayer Technologies, Inc., "Build Your Own Cloud Server," http://www.softlayer.com/cloudlayer/build-your-own-cloud.

[18] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower'10, 2010, pp. 1–8.

[19] VMware, "VMware vCenter Server," http://www.vmware.com/products/vcenter-server.

[20] "Why has CPU frequency ceased to grow? ," https://software.intel.com/en-us/blogs/2014/02/19/why-has-cpu-frequency-ceased-to-grow, 2014.

[21] Microsoft, Inc., "Pricing Calculator — Windows Azure," http://www.windowsazure.com/en-us/pricing/calculator/?scenario=virtual-machines.

[22] J. Walrand, *An Introduction to Queueing Networks*. Prentice Hall, 1988.

[23] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, Model-driven Autoscaling for Cloud Applications," in *Proceedings of the 11th International Conference on Autonomic Computing*, Philadelphia, PA, USA, 2014.

[24] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[25] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *Transactions on Computer Systems*, vol. 30, no. 14, 2012.

[26] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz, "NapSAC: Design and implementation of a power-proportional web cluster," in *Proceedings of the 1st ACM SIGCOMM Workshop on Green Networking*, New Delhi, India, 2010, pp. 15–22.

[27] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," in *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Banff, Alberta, Canada, 2005, pp. 291–302.

[28] ProfitBricks GmbH, "Live Vertical Scaling," PROFITBRICKS IAAS, Tech. Rep., 2012.

[29] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and Self-configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters," in *Proceedings of the 6th International Conference on Autonomic Computing*, Barcelona, Spain, 2009, pp. 117–126.

[30] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas, "Nobody Ever Got Fired for Using Hadoop on a Cluster," in *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing*, Bern, Switzerland, 2012, pp. 2:1–2:5.

[31] Gigaspaces Resource Center, "Scale Up vs. Scale Out," http://www.gigaspaces.com/WhitePapers, 2011.

[32] M. Sevilla, I. Nassi, K. Ioannidou, S. Brandt, and C. Maltzahn, "A Framework for an In-depth Comparison of Scale-up and Scale-out," in *Proceedings of the 2013 International Workshop on Data-Intensive Scalable Computing Systems*, Denver, CO, USA, 2013, pp. 13–18.

[33] M. Schwarzkopf, D. G. Murray, and S. Hand, "The Seven Deadly Sins of Cloud Computing Research," in *Proceedings of the 4th USENIX Conference on Hot Topics in Cloud Computing*, Boston, MA, USA, 2012.

[34] W. Iqbal, M. N. Dailey, and D. Carrera, "SLA-Driven Dynamic Resource Management for Multi-tier Web Applications in a Cloud," in *Proceedings of the 10th International Symposium on Cluster, Cloud and Grid Computing*, Melbourne, Victoria, Australia, 2010, pp. 832–837.

[35] M. Sedaghat, F. Hernandez-Rodriguez, and E. Elmroth, "A Virtual Machine Re-packing Approach to the Horizontal vs. Vertical Elasticity Trade-off for Cloud Autoscaling," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, Miami, FL, USA, 2013, pp. 6:1–6:10.

[36] N. Bonvin, T. Papaioannou, and K. Aberer, "Autonomic SLA-Driven Provisioning for Cloud Applications," in *Proceedings of the 11th International Symposium on Cluster, Cloud and Grid Computing*, Newport Beach, CA, USA, May 2011, pp. 434–443.

[37] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically Scaling Applications in the Cloud," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 45–52, Jan. 2011.

[38] M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski, "Scale-up x Scale-out: A Case Study using Nutch/Lucene," in *Proceedings of the 2007 International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, 2007, pp. 1–8.

[39] H. Yu, J. Moreira, P. Dube, I.-H. Chung, and L. Zhang, "Performance Studies of a WebSphere Application, Trade, in Scale-out and Scale-up Environments," in *Proceedings of the 2007 International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, 2007, pp. 1–8.

[40] P. Brebner and J. Gosper, "How Scalable is J2EE Technology?" *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 3, pp. 4–4, 2003.

[41] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs Scale-out for Hadoop: Time to Rethink?" in *Proceedings of the 4th Annual Symposium on Cloud Computing*, Santa Clara, CA, USA, 2013, pp. 20:1–20:13.