Right-sizing Server Capacity Headroom for Global Online Services

Citation for published version:
Verbowski, C, Costa, P, Leather, H, Franke, B & Thayer, E 2018, Right-sizing Server Capacity Headroom for Global Online Services. in 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). IEEE, Vienna, Austria, pp. 645-659, 38th IEEE International Conference on Distributed Computing Systems, Vienna, Austria, 2/07/18. DOI: 10.1109/ICDCS.2018.00069

Digital Object Identifier (DOI):
10.1109/ICDCS.2018.00069

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)

Right-sizing Server Capacity Headroom for Global Online Services

Chad Verbowski
University of Edinburgh
c.e.verbowski@sms.ed.ac.uk

Ed Thayer, Paolo Costa
Microsoft
edcthayer@gmail.com, paolo.costa@microsoft.com

Hugh Leather, Bjoern Franke
University of Edinburgh
hleather, bfranke@inf.ed.ac.uk

***Abstract*—We present a capacity planning case study showing a significant opportunity for improving the utilization of a large, low-latency, highly available online service containing 100K+ servers spanning 9 geographic regions. Analyzing 30 PB of traces over 90 days we devised a new iterative black-box capacity planning model using the discovered relationships between workload, utilization, and quality. We verified the model on 1,000s of servers showing capacity reductions between 20% and 40% with effectively no impact on workload latency, availability, or the capacity required for disaster recovery. These results are confirmed experimentally by shrinking production server pools to cause the remaining servers to run at higher utilization, and using data from real-world large scale unplanned failures. Finally, we show examples of using our model for offline regression analysis to detect critical issues before their deployment.**

I. Introduction

A growing number of large online services, including search, social networking, online shopping, media services, gaming, and email, can require up to 1M servers [1]–[3] in globally distributed datacenters to provide low latency and high availability to customers around the world [4]. Since the cost of such services can exceed US$1 billion annually [5] there is a significant financial and environmental impact if they are inefficiently operated [6]. Improving efficiency through reducing excess capacity by as little as 1% can save millions of dollars per service.

To gauge the excess capacity common in the industry we analyzed traces from more than 100K servers in a large online service. Our analysis confirmed previous observations [7], [8] indicating a significant opportunity for capacity savings by showing at least half of all global resources are idle at any given time. However, we found diurnal global online service workloads [9] cause individual datacenters to periodically run out of capacity while datacenters on the opposite side of the world are underutilized. Although this available capacity can be used for offline workloads without latency requirements [10]–[14], it is unsuitable for online request processing as it would increase response latency, and changes of even 100ms decrease user engagement [15].

Current capacity planning approaches either forecast capacity requirements using a queuing theory based model [16]–[18] of the system (hardware, software, network) and workload [19], [20], or dynamically modify capacity allocations based on real-time feedback of resource usage and quality of service (QoS). In large scale services, the queuing model is impractical to maintain given the scale of elements and their change rate. Models must consider continual changes [21], [22] from 10Ks of engineers changing software with 10Ms of code lines running on 100Ks of servers, each with 70K files, 200K settings with 16M to 68M daily reads and 0.6M to 2.9M daily updates [23]. In practice, models based on simplified assumptions are either inaccurate, or are quickly invalidated as the system evolves.

Dynamically allocating capacity [24]–[28] avoids the complexity of building and maintaining models by treating the system as a black-box and modifying capacity based on forecast changes in resource usage. This avoids the modeling approach overhead, and can automatically adapt to changes in demand or failures. However it is unsuitable for large online services for three reasons. First, the diurnal variation in capacity requirements for a given datacenter can be 1,000s of servers, which is more than is readily available to dynamically allocate during peak demand.

Second, prior work underestimated the time required to change the capacity of a system. For example, it does not account for service start-up times in the minutes caused by large in-memory stores required to load state into memory, managed code applications requiring minutes of JIT compilation, or services requiring cache priming. Furthermore changing capacity by 1,000s of servers happens in weeks or months rather than seconds because of the logistics involved along with the cost and service impact of changing it more frequently. This is sub-optimal because it requires excess capacity, called headroom, to accommodate any planned maintenance, unplanned failures, or unexpected workload increases while large capacity changes are made.

Third, this approach cannot easily predict capacity requirements of a new or modified system without deploying it to production. For example, it overestimates capacity when historical trends vary or future workload volume and processing costs are inaccurate because the business impact of underestimating can be severe.

In contrast to these approaches, we propose a black-box capacity planning approach requiring only workload, resource usage, and QoS measurements. We argue that only the service's external response characteristics are necessary to achieve an accurate forecast, avoiding the costly iterations required in the dynamic capacity approach, and avoiding the complexity and difficulty of accurately modeling the service's software, configuration, hardware, and network.

Our approach is possible because the industry is moving towards the micro-service [29] design pattern to exploit data parallelism. This differs from the traditional approach of having pools[1] of enterprise grade servers running multiple workloads [30], to a model where services are composed of simpler micro-services each running in their own pool of consumer grade servers. A pool per micro-service makes it easy to measure and optimize resource usage because there is a single workload on each server. A micro-service instance per server avoids resource contention across instances. Furthermore, this approach can be used for offline regression analysis of resource requirements and QoS impact before deploying micro-service changes.

Despite the apparent simplicity of our approach, there are three subtle complexities that must be addressed. First, ensuring sufficient server metric granularity to enable isolation of the workloads we are capacity planning for. This requires accurately instrumenting each major server workload to ensure noise from background workloads (such as system processes) does not impact the resource usage measurements for the workload we are planning for. This is difficult as there is no standard definition for a unit of workload within a service, and it can be challenging to keep track of resource usage per unit of workload.

The second challenge is identifying the non-linear relationship between workload and QoS. In practice, there is no standard QoS metric, each service defines it using metrics such as latency, availability, and reliability of a workload response. The impact of resource limiting on QoS is non-linear, reflecting the software, hardware, and networking complexity of the service implementation. Since only the black-box relationship between workload and QoS is needed, we avoid the complexity and fragility of creating and maintaining a queuing theory model. Instead we create an equation for this relationship using simple curve fitting on historical data, and if needed, conduct a minimal set of production experiments to obtain additional data points.

Because servers in a pool may behave differently, the third challenge is to identify groups of servers within each pool with a similar profile of QoS, limiting resource, and workload. In practice, we were able to train a decision tree to identify the optimal server groups in each pool with minimal complexity and high accuracy.

We validated our approach on a large service and found 31% of servers could be removed without impacting QoS. Since testing our approach at full scale was impractical, as the servers belong to different organizations, we verified our results on a smaller yet representative subset of servers (1000s). We found the approach coped with the complexity found in our large service naturally and intuitively, enabling significant capacity savings. It enabled the development of tools for managing the capacity implication of all changes before they are deployed, ensuring efficient ongoing capacity utilization. We believe our approach is generally applicable to large scale low-latency services with diurnal workloads developed using a micro-service design pattern [31], [32].

The main contributions of this paper are:

1) A new capacity allocation methodology providing significant savings and addressing practical challenges overlooked by other methods.
2) Show capacity savings of between 20% and 40%, confirmed on a large production service.
3) Detailed resource usage analysis of a service spanning 100,000s of servers[2] spanning 9 datacenters over 90 days showing:
    a) Well-managed servers use only 2% downtime, yet 17% was the observed average.
    b) CPU usage averaged 23% for the servers studied, with 80% using less than 30% CPU.
    c) CPU usage spikes are rare, only 15% of servers had a CPU spike larger than 40%.

II. Methodology

Our approach forecasts the QoS impact of changing server allocations for a specified workload volume. For example, reducing QoS requirements by 5 ms may require 10% less services. The QoS requirement for each micro-service is defined as a set of Service Level Objectives (SLOs). Each SLO is a specific metric and the minimum threshold of their values. For example, response latency must be less than 500ms, and reliability must be 99.999%. It is possible for every micro-service to have a unique combination of metrics and thresholds for their QoS requirement, however in practice these are typically defined for the overall service to capture the intended QoS of the end-to-end service. Capacity planners use this in conjunction with workload trends, expected failure rates, and QoS business requirements to determine how many servers are needed.

Our goal is to be practical and realistic by avoiding the complexity, fragility, and service specific aspects of the modeling approach, yet retain the deterministic understanding between QoS, workload, and resource usage it provides. To enable broad adoption it must be generally applicable to services in diverse environments and avoid service specific customization. It needs to enable offline 'what-if' regression analysis of changes to determine their capacity and QoS consequences. It needs to err on over allocating capacity to avoid the business impact [33]–[35] of low QoS from running out of capacity. Services typically require between 99.95% [36] and 99.999+% [37] availability with peak

[1] A server pool is a set of servers with a network load-balancer distributing incoming requests evenly across them. All servers have the same software and hardware.

[2] For confidentiality, we omit certain information, e.g., the absolute number of servers, focusing instead on a detailed subset of the server fleet.

workload despite portions of the system being offline due to unexpected failures.

We require the following conditions to be true:

A predictable relationship between workload, resource usage, and QoS: Large complex services with varied resource requirements are created as a set of micro-services running on their own pools because it enables their resource requirements to change without impacting the others, and enables scaling with demand by changing how many servers are in each pool. Absent this design micro-services would not be able to use all resources on a single server, and competition between micro-services for the same resource must be managed as part of capacity planning.

Service composed of micro-services in server pools: many small servers are preferable over fewer large servers to minimize wasted resources and maximize available capacity when individual servers fail. The minimal size of a server is determined by the maximum resources required for a micro-service to cost effectively process a request with the required QoS. Micro-services are deployed to production on pools of identical servers (such as cloud VMs [38]), each running the same code, with a load-balancer evenly distributing the incoming workload across the pool of servers. The uniformity of servers in the pool enables the micro-service to remain available during planned or unplanned outages of individual servers by amortizing requests over the remaining servers in the pool. Micro-service capacity is specified at server granularity by modifying pool allocations.

Our methodology has four steps for identifying the optimal capacity of an existing system, and then for performing ongoing regression analysis to ensure optimal capacity is maintained when the configuration, hardware, software, or network, changes.

Step 1 Measure: Confirm metric accuracy and group servers with similar workload, resource usage, and QoS.

Step 2 Optimize: Determine the minimal servers for each pool to operate within its QoS using historical and experimental data to forecast the results.

Next we describe our methodology for enabling ongoing offline regression analysis of changes to maintain optimal capacity efficiency.

Step 3 Model: We create a synthetic workload to drive an offline system with the same response characteristics as a production workload. For example, the synthetic workload may need to mock [39] incoming requests or responses from calls to dependent services with a diversity of requests and responses matching those observed in production. This is required when user specific data such as mailboxes are accessed, incoming requests have user credentials, or QoS and resource usage is proportional to the diversity of incoming requests.

Step 4 Validate: Validation of changes using an offline stress-testing process to measure the workload, resource usage, QoS profile of the system given a range of synthetic workloads. The profile is then compared to the system profile before the change to determine the impact.

Micro Service	**Description**
A	In-Memory Storage (similar to MemCached [40])
B	Modifies incoming requests such as spelling corrections.
C	Orchestrates a workflow of stateless processing modules.
D	Converts responses from data to formatted web pages.
E	Spit-TCP proxy, CDN, load balancer, and authentication service (similar to Squid)
F	In-Memory storage with custom processing logic.
G	High volume, low latency, metrics collection system used for automated operational decisions.

Table I: Description of micro-services running in server pools for our analysis.

Fig. 1 shows an overview of these steps and the remaining subsections describe each step in detail. Academic work has assumed things like metrics are in place and workload is reproducible and re-playable. These are not valid assumptions in practice, so our methodology includes step 1 for validating our metrics for online capacity planning, and step 3 for verifying our synthetic re-playable workload for offline regression analysis.

A. Measuring Resource Usage

Our approach is based on the ability to accurately measure changes in workload and the corresponding resource usage and QoS offered. In practice resource usage metrics are not partitioned by workloads such as background server tasks and the primary micro-service workload and are often measured independently with respect to time such as the standard CPU, IO, Network metrics exposed by the server operating system. This lack of accuracy makes the relationship between workload and resource usage appear random due to instantaneous variations in workload, variations in distributing requests across servers in the pool, and variations in request processing cost. Identifying these per-workload metrics is the only system dependent aspect of our approach, and is the only step which may require modification of the system to expose new metrics.

For the micro-services we analyzed (see Table I) we found they followed the common design pattern of isolation along standard operating system boundaries such as processes, enabling us to measure our primary resource usage by simply filtering by process names. In other cases we added new metrics to the systems we investigated. Section II-A1 describes our approach for confirming the right metrics are in place, by iterating on them until we find a linear relationship between a constrained resource and workload.

Once we have confidence in our metrics, we identify groups of servers performing the same task with the same workload to resource usage response. Capacity planning is then accomplished by determining the QoS impact of adding or removing servers in each group. Section II-A2 describes

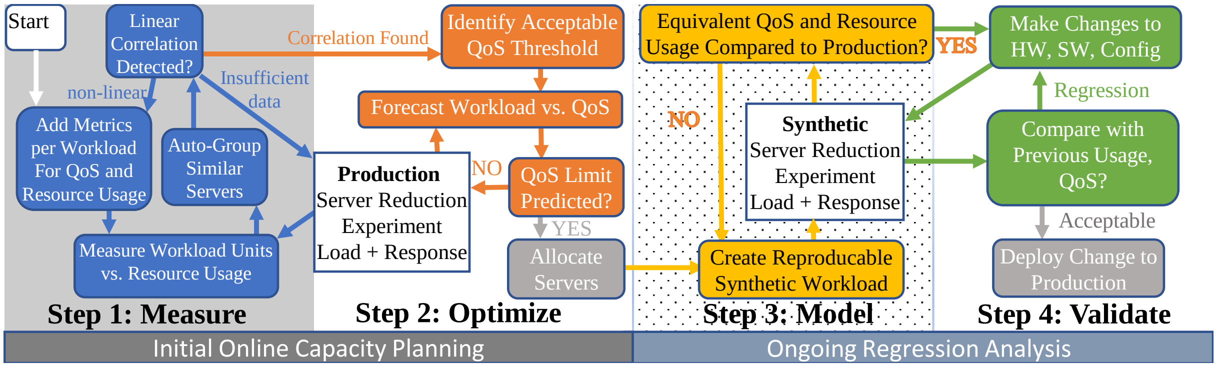
Start
Linear Correlation Detected?
Correlation Found
Identify Acceptable QoS Threshold
Equivalent QoS and Resource Usage Compared to Production?
YES
Make Changes to HW, SW, Config
non-linear
Insufficient data
Forecast Workload vs. QoS
NO
Synthetic Server Reduction Experiment Load + Response
Regression
Add Metrics per Workload For QoS and Resource Usage
Auto-Group Similar Servers
Production Server Reduction Experiment Load + Response
NO
QoS Limit Predicted?
Compare with Previous Usage, QoS?
YES
Acceptable
Measure Workload Units vs. Resource Usage
Allocate Servers
Create Reproducible Synthetic Workload
Deploy Change to Production
Step 1: Measure
Step 2: Optimize
Step 3: Model
Step 4: Validate
Initial Online Capacity Planning
Ongoing Regression Analysis

Figure 1: Our methodology steps for optimizing existing systems and offline regression analysis for new changes.

our approach for identifying these groups, showing it is possible to train a decision tree to do this with minimal complexity and high accuracy.

1) Workload Metric Validation: Our work requires reliable measurement of the incoming workload, associated resource usage, and resulting QoS. To ensure existing metrics are sufficiently accurate, or to validate the accuracy of metrics added to existing service for this purpose, we developed a pragmatic approach for validation. We assume proper workload metrics have a tight linear correlation between units of work and increases in their primary limiting resource, e.g., CPU increasing linearly with requests volume. If the metric does not correlate well with the limiting resource then we likely failed to accurately capture the resources used to process a request. We use this validation in a feedback loop, until an accurate result is obtained. For example, when analyzing a micro-service similar to MemCached [40], we found the metric was noisy because the workload was measuring requests to multiple tables. After splitting workload into two metrics for each table, both exhibited a linear relationship with CPU. In another case periodic resource spikes correlated with log uploads of many GB / hour. Our technique discovered this anomaly and enabled removing its effect. We found that in practice there was virtually no need to add new code to collect new metrics to measure workload or QoS, we simply needed to identify which instances of the existing ones to use.

To illustrate this approach, we show the workload metrics versus each resource (Processor, Disk, Memory, Network) for a micro-service in Fig. 2. We measure requests per second (RPS) on the x axes and the consumption of various resources on the y axes. This data was gathered over one day of usage on micro-service D[3] running on separate pools in 6 data centers, and comprise queue length, rates, errors, and usage of each resource. We see CPU shows a linear relationship with little variance across a range of RPS, indicating for this service, RPS is a sufficiently accurate workload metric. We believe a metric is accurately isolated per workload when it has low variance as seen for processor utilization. High variance typically indicates noise from other workloads, or another resource is the limiting one. For example, networking counters show a linear relationship characteristic, and we see more variation of bytes and packets across data centers. Disk bytes read and memory paging look very similar, indicating most disk activity is likely due to paging on the system, suggesting the application is not using much IO. The vertical patterns indicate a large variation in memory and disk activity for a given workload rate. Error counters and queues for these resources are static in the steady-state and are more suitable for anomaly detection.

The micro-services we analyzed were designed to have CPU as the limiting resource, which our results confirmed. We believe this is common for most large online services because CPU is the most expensive server resource. For example modern data center networks have much cheaper costs per server and networks are not typically a bottleneck at the current 40Gbps server inter-connectivity and are less likely to be with the upcoming evolution to 100Gbps. The growth rate of RAM size, SSD size, and network bandwidth per unit of cost is higher compared to the CPU. CPU demand by micro-services has grown due to increasing requirements for encrypt all data stored, transmitted, and held in memory. Even if the cost dynamics or software requirements change in the future, we believe in general it is unnecessary to focus on all resources for capacity planning, and if needed our approach can be directly applied to discover and use the new limiting resource.

Once we are confident of our metrics at a per server level we identify the groups of servers with the same request inputs having similar workload to resource usage response. These are the groups we plan capacity for, by determining the QoS impact of adding or removing servers from the group. Section II-A2 describes our approach for identifying these groups, showing it is possible to train a decision tree to do this with minimal complexity and high accuracy.

[3] Micro-service D creates consumer web pages from data obtained from other micro-services

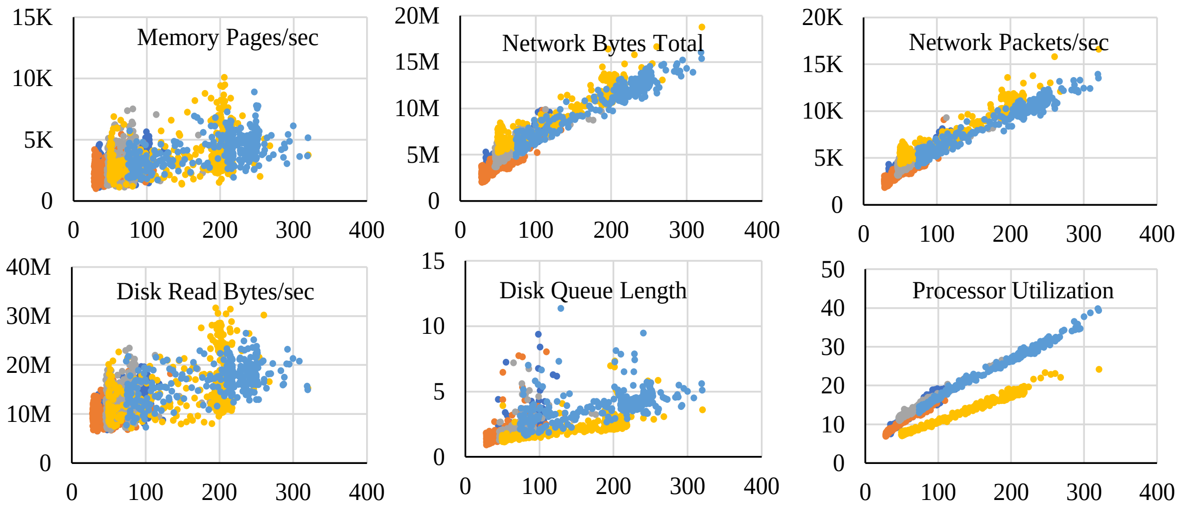

Figure 2: Various resource counters versus workload. Each point is the 1-minute average across servers in a the micro-service D pool. Each shape is a datacenter, the x-axis is always workload (RPS), the title defines the y-axis.

2) Identifying Capacity Planning Server Groups: Capacity is managed by adding or removing servers from pools dedicated to a micro-service where each server receives an equal portion of the incoming workload to the pool and produces an identical response. This makes it natural to plan capacity for each micro-service's pool. However, we find there can be slight differences across servers in the pool such as minor hardware variations or non-uniform communication between servers from minor workload processing differences such as some performing extra tasks as part of being a primary owner of a replicated data set. For this reason, we may need to partition pools into smaller groups of servers exhibiting the same resource usage and QoS response for an equivalent workload.

To illustrate the similarity of servers within a pool we use a scatter plot of minimum[4] and maximum CPU for each server in a pool over a period spanning the expected time range of workload volume (typically a day for diurnal workloads). To confirm we have considered a full cycle of normal operations we simply expand the range of data considered until the resulting scatter plot no longer changes. Fig. 3 shows tight clusters of servers in each datacenter, indicating a consistent daily upper and lower bound. Interestingly, one of the pools shown has servers operating in two distinct clusters, one with a lower minimum and maximum compared to the other. Investigating this we found all servers in the less utilized range are newer and more powerful than the other. This suggests the technique can be useful for quickly evaluating the impact of hardware or software changes on server utilization.

To automate the analysis for identifying the groups of servers within pools in addition to the clustering approach results shown inFig. 3, we used a decision tree based feature vector approach. Each server's feature vector contains:

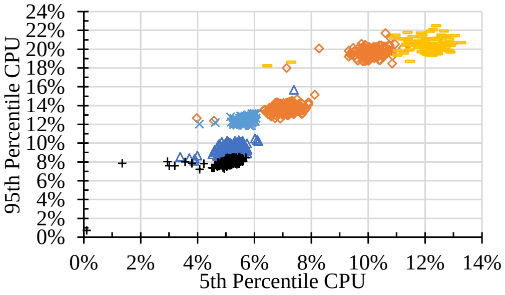

Figure 3: Scatter plot of the high and low CPU usage for each server in pool I. Shapes are datacenters.

1) The 5th, 25th, 50th, 75th, 95th percentile CPU utilization values for each server in the pool.
2) The slope, intercept, and R^2 value estimated by a linear regression on each pool across

$$(p_i, c_i) : i \in \{1, 2, ...server\ count\}$$
$$p_i \in \{5, 25, 50, 75, 95\}$$
$$c_i = CPU\ utilization\ for\ percentile\ p_i$$

We trained a decision tree with 5 fold cross validation with manually labeled pools using a minimum leaf size of 2000 machines. The tree contained 34 splits, achieving an R^2 = 0.746. The area under curve (AUC) for the Yes and No prediction probability is 0.9804.

Using this technique we found 55% of pools with a typical diurnal workload also exhibited a tightly bound CPU utilization range, suggesting a predictable relationship between CPU and workload. Investigation of the remaining

[4] Using the industry best practice of 5th percentile to represent the minimum and the 95th percentile to represent the maximum eliminates outliers caused by rare hardware failures or other anomalies.

pools found they were running multiple workloads, typically background administrative tasks, and could fit our analysis when the resource usage of these workloads was identified and independently modeled. Based on this, we expect most pools will exhibit a predictable workload to resource utilization relationship.

B. Optimizing Server Pool Capacity

QoS requirements for micro-services limit the minimal capacity required for a pool. QoS is typically defined in terms of an acceptable failure rate and request latency threshold. Through capacity forecasting business owners can determine these thresholds, by trading off the impact of lower success or performance with business value.

While identifying the relationship between workload and resource usage ensures we have accurate metrics for understanding the system, we determine the minimal resources required using our measurement of the relationship between workload and QoS. Ideally enough historical data exists for each micro-service to understand the relationship between workload and QoS. Section II-B1 describes our analysis of some micro-service server groups where unplanned capacity events ('natural experiments') caused higher than normal workloads, providing us with additional data to perform our capacity optimization.

In cases where insufficient data exists we conduct experiments removing servers from production pools to increase workload on the remaining servers while measuring their QoS response. These experiments are expensive to perform because of the potential unknown QoS impact to production requests. For this reason, experiments are manually supervised by service operators who can quickly restore capacity if the QoS is to low. The overhead of business risk and human cost constrain the size of each reduction experiment and limits the total number of experiments. Section II-B2 describes our approach to performing these experiments with minimal business risk and iterations.

1) Capacity Planning using Natural Experiments: One goal of capacity planning is to identify the additional resources required to maintain QoS during unplanned capacity loss or increases in workload. To determine if our model of the relationship between resource usage and workload holds during such events we analyzed the data from two historical unplanned capacity impacting events.

The first event spanned two hours causing pools in multiple datacenters to receive a median 56% increase in workload volume as shown in Fig. 4, with one datacenter receiving an increase of 127%. Fig. 5 shows each datacenter's CPU usage followed the predicted linear relationship during the previous or subsequent 2 days. During this period latency remained below 26ms indicating no QoS impact.

The second event (Fig. 6) shows the latency of a pool during a separate unplanned event with 4 times the normal traffic volume. The trend line indicates there should be a predictable relationship between latency and workload. Each point in the chart represents a sample from each server within the pool. The shape represents the datacenter location of the pool. Note, the elevated latency at low workload is typical, and caused by additional work performed when the software starts such as priming caches and pre-compiling managed code.

Our analysis of the first event confirmed the results we obtained by manually removing servers to increase workload. The second enabled us to obtain data at much higher workloads than we were comfortable obtaining experimentally and discover how the trend line would behave beyond the workloads we could accurately predict through extrapolation. We believe that analyzing the effect of unplanned events is a useful way to learn more about the characteristics of the system, and if there is sufficient data from these there may be no need to experiment to discover this information.

2) Capacity Planning through Experimentation: We use response surface methodology (RSM) [41] to explore the relationships between several explanatory variables (workload) and one or more response variables (QoS and resource usage), using a sequence of designed experiments to obtain an optimal response. RSM is ideal because it enables us to use historical data and natural experiments, and when additional data is needed we can run experiments to until we reach our tolerance for QoS impact. We do not conduct experiments to fully explore the relationship between QoS and further pool reduction due to the potential impact to customers. Instead we curve fit across the historical and RSM experimental data to forecast the overall relationship between QoS and pool size, and use this to make capacity allocation decisions meeting business QoS needs. The RSM process starts by identifying the factors affecting the response variables, and then iterates over the following steps:

1) *Model*: Model the experimental data to determine gradient along which the objective function changes in the desired direction, and
2) *Extrapolate*: Extrapolate along the model's gradient to identify the next center point for collecting experimental data and perform the next iteration.

Before applying RSM we applied the analysis described in Section II-A1 to historical pool data to identify where negative correlation exists between the number of servers processing traffic and the CPU utilization after controlling for total datacenter load. We then examine the relationship between the average number of servers and average CPU utilization for a single pool across a long period of historical data, typically months. Since the total workload for a micro-service is distributed equally across all servers in the pool, the total workload is used to partition historical time points when the pool's servers had comparable loads. This increases the number of observations included in each fit while increasing the noise within each partition since a narrow range of total workload is present in each partition.

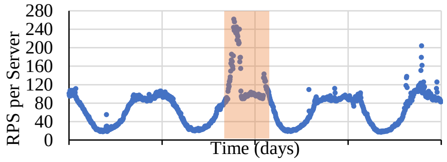

Figure 4: The time series of the pool workload spanning 2 days before and after the event.

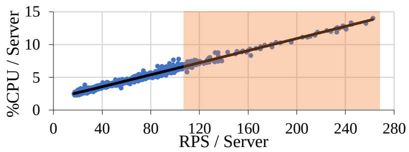

Figure 5: CPU vs. the workload RPS over the period spanning the event. We see the large increase in workload provided data confirming the CPU load prediction.

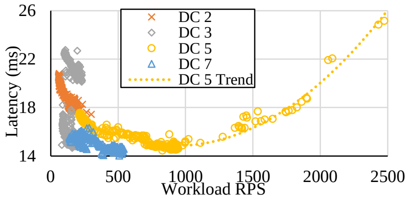

Figure 6: Daily traffic for 5 datacenters showing DC 5 behaving as predicted when receiving 4x more requests during the unplanned event.

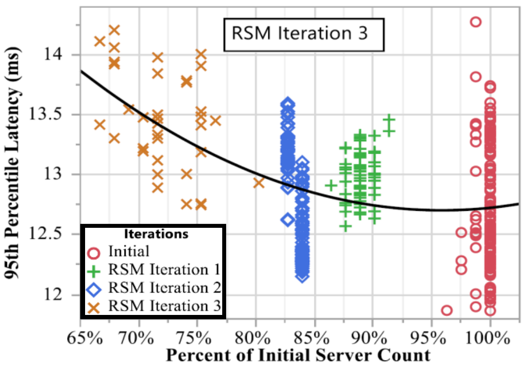

Figure 7: RSM experiment iterations, showing latency increases from successive server reductions until 14ms QoS limit is reached.

In practice, we could identify partitions such that the first order fit values are not overwhelmed by this noise.

An iterative RSM approach is used to experimentally change the number of servers used by a pool while measuring the corresponding QoS, and then using this result to forecast the QoS impact of further reductions. Production experiments often have additional natural changes in server counts resulting from normal service operations (deployments, traffic load shifts, etc.), as seen in 3rd iteration of the chart in Fig. 7. Our experimental design controls for total pool workload since we are modeling how pool QoS changes as a function of the number of servers processing a given total workload.

Micro-service pools can exhibit different performance characteristics in each datacenter so we index both the datacenter ($d \in \{1,2,\ldots,D\}$) and the pool ($i \in \{1,2,\ldots,I\}$) being optimized in our collected data. For pool s_{id}, let $\{r_{idj} : j = 1,2,\ldots,J_{id}\}$ be the observed partitions of total pool workload i in datacenter d. While the range of workload is organic and hence not controllable, working directly with a pool owner we identify J_{id} to ensure sufficient data is available within each heavily used partition to generate reasonable model fits for identifying the objective function's gradient. Once defined, the total load partition $\{r_{idj}\}$ partitions the time points for data collecting, and we denote the set of time sets for s_{id}s total load is in r_{idj} as $\{t_{idj} : j = 1,2,\ldots,J_{id}\}$. Specifically, t_{idj} is the set of times when pool i's total workload falls into partition j at datacenter d. Note, we use $|t_{idj}|$ as the cardinality of set t_{idj}. Since the server count varies with time (naturally) and is our experimental control variable, we observe the average number of servers used at each time within t_{idj} and we denote these observed counts as $\{n_{idjk} : j = 1,2,\ldots,J_{id}, k = 1,2,\ldots,|t_{idj}|\}$. Lastly, we compute the average observed latency value (percentile determined by the business) $\{l_{idjk} : j = 1,2,\ldots J_{id}, k = 1,2,\ldots,|t_{idj}|\}$ for pool i in datacenter d when its total workload is in partition r_{idj} at the k^{th} time in t_{idj}.

In the experiment step of the data collection process, each pool reduces their server count in a datacenter for a period of (roughly) one week. Latency data is collected from these experimental times, added to the historical data for the pool, and a second order quadratic polynomial is fit for each total load partition r_{idj}. Specifically,

$$\tilde{l_{idj.}} \approx a_{idj,2} n_{idj.}^2 + a_{idj,1} n_{idj.} + a_{idj,0} n_{idj.} \qquad (1)$$

where the model parameters $a_{idj,2}, a_{idj,1}, a_{idj,0}$ are estimated using robust regressions (RANSAC) [42] using observed data l_{idjk}, n_{idjk} for pool i in datacenter d while total workload falls into partition j. Fig. 7 shows the results from multiple iterations for a typical micro-service.

C. Synthetic Workload Modeling

A key benefit of our approach is the ability to validate changes to QoS and resource usage for a given workload. This enables us to avoid allocating excess capacity to compensate for potential capacity or QoS regressions accidentally being deployed to production. To enable this (step 3 in our methodology from Fig. 1) we first verify our synthetically produced workload causes the same QoS and resource usage relationship we observe in our measurements of production server pools. For the same volume of synthetic workload we see the same QoS and resource usage values.

This is a novel and important step because it ensures we have an accurate distribution of request variations, and an appropriate distribution of responses from any remote call dependencies made outside of the micro-service we are validating. Without matching the synthetic workloads to the production workload, it would only be possible to detect a change in capacity or latency had happened, but it would not be able to accurately determine the magnitude. Once this is complete, we have established our baseline for comparing any changes we make to the system.

D. Offline Capacity Validation

As described in Fig. 1 step 4, before making any change to a production micro-service group of servers we first make this change to the offline version and run the synthetic workload to measure the QoS and resource usage response for a wide range of workload volume. By comparing these with our baseline values, we can determine the impact to capacity or QoS before any change in hardware, software, configuration, or networking, is deployed to production. Most importantly we not only detect when a change happens, we also determine the curve describing the change, enabling adjustment of capacity plans if needed. Furthermore this curve tells us what we expect the QoS (performance) and resource usage of a software change will be in production, before we deploy it. This enables us to adjust production server pool capacity for new features or changes if needed before the change is deployed.

Using this approach for each version of software released to a server pool, we detect scalability problems that would have caused unplanned service outages before they were released to production. Our system uses two server pools of the same size and hardware, one running with the change and the other without. We precisely generate identical workloads to each pool enabling us to detect changes with high confidence and precision. We make small workload increments over time to obtain a broad set of data for latency and resource utilization. Finally, we compare the pool results to understand the impact of the change.

III. Method Evaluation

We validate our approach by predicting the QoS (latency) impact of removing servers from two micro-service pools containing 100s of servers and then experimentally measuring the impact, as described in Section III-A. For the first service we predicted the 95th-perc. latency to be 31.5ms after removing 30% of servers and measured this to be 30.9, and for the second one we predicted it to be 52.6ms after removing 10% of servers and measured it as 50.7ms.

We also analyzed the capacity usage of all production servers in a large online service to understand the opportunity for capacity savings. Our results, described in Section III-B, show a significant opportunity for savings with 80% of servers using less than 30% of their bottleneck resource, CPU. Next we analyzed the availability of these servers in Section III-B2 to determine the capacity used to compensate for planned and unplanned maintenance. We found only 2% of capacity is needed, and a potential capacity savings of 15% of total capacity if some micro-services improved their planned maintenance practices. These results are based on our analysis of 30 PB of data containing performance counters and request processing data spanning 90 days from hundreds of thousands of production servers. Production servers are a subset of all servers containing only those used in the real-time processing of service requests. Performance counters were sampled every 100 ns [43] and averaged over a 120 s window. The window size was selected to be as large as possible to minimize the cost of storage and the overhead of collecting approximately 3 GB/s.

	RPS / Server Percentiles		
Experiment Stage	50%	75%	95%
Original Server Count	249.5	309.3	376.8
30% Server Reduction	390.4	461.1	540.3
% Change	56%	49%	43%

Table II: Lists 50th, 75th, and 95th percentiles of RPS/Server during the two stages of the server pool D reduction experiment. The count of active servers during each stage agreed with the expected 30% reduction, but the RPS/server increased by more than 43% at the 95th percentile due to increased traffic load to the service during the experiment.

A. Evaluation of Capacity Forecasting

Since our forecasts are based on extrapolations there is no way of knowing how the shape of these trend curves will shift, it is best to remove servers slowly and monitor the accuracy of these forecasts. More data closer to the target performance points would improve the later model accuracy. Section III-A1 and III-A2 describe the two server pool reduction experiments.

By comparing the resource utilization at varying workloads for each version of software released to a new server pool, we were able to detect scalability problems that would have caused unplanned outages to the online service before they were released to production.

1) Detailed Analysis of the Server Pool B: The first server count reduction was performed for a query modification service (Server Pool B in data center DC 1). The original server pool servers were observed over 5 weekdays to be processing 377 requests per second (RPS) at the 95th percentile of load (see Table II). Fig. 8 and Fig. 9 show the average total percent processor time and latency at these loads was 12% and 30.5ms.

During the second stage of the experiment, the request volume per server was increased by reducing the server pool size by 30%. Coincident with this reduction, the production traffic increased to this service resulting in 540 RPS/server (43% increase) at the 95th-perc. of load.

Fig. 8 overlays the average processor percentages/server from both experimental stages. A linear model trained on

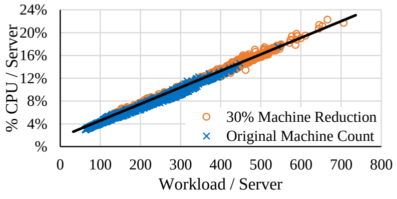

Figure 8: Average %CPU versus average workload per server for server pool B in datacenter 1 follows our linear prediction.

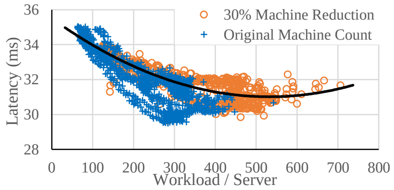

Figure 9: Average 95th percentile latency versus average workload per server for server pool B in datacenter 1 follows our quadratic prediction.

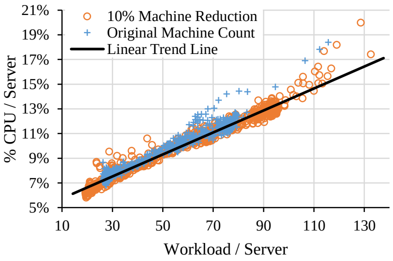

Figure 10: Average %CPU versus average workload per server for server pool D in datacenter 1 follows our linear prediction.

the original server pool size

$$y = 0.028 * RPS + 1.37 \quad R^2 = 0.984 \quad N = 1221$$

would have predicted an average 16.5% processor utilization/server at an average 540 RPS/server. Performance during the reduced server stage exceeded the forecast value coming in at 17.4%

$$y = 0.029 * RPS + 1.7 \quad R^2 = 0.99 \quad N = 576$$

The forecast error resulted from the increased intercept, which historically has coincided with code or data deployments. The scatter plot in Fig. 9 shows how the average 95th percentile of internal latency changed across the two experiment stages. A quadratic polynomial trained on the original server pool size

$$y = 4.028e^{-5} * RPS^2 - 0.031 * RPS + 36.68$$
$$R^2 = 0.79 \quad N = 1221$$

forecast an average latency of 31.5ms, while the measured average latency was 30.9ms. Data is insufficient to forecast when the latency curve will rise at even higher loads.

Since we do not know the underlying model for the system we are analyzing, our analysis techniques did not assume the shape of the underlying data distribution. We started by trying the simplest techniques first and found that quadratic polynomials worked in this case and for 10s of other server pools, so there was no need to evaluate more complex approaches such as a Gaussian model. Techniques such as nearest neighbor are useful for interpolation and not accurate for the extrapolation we need to forecast resource usage and latency at higher workloads.

2) *Detailed Analysis of Server Pool D:* The second server count reduction experiment was performed for a service responsible for routing traffic within the data center. The original server pool servers were observed to be processing 78 workload units at the 95th percentile of load (see Table III). Fig. 10 and Fig. 11 show the average processor and 95th percentile internal latency at the 95th percentile of load to be 12.1% and 52.8ms.

	RPS / Server Percentiles		
Experiment Stage	50%	75%	95%
Original Server Count	56.8	74.8	77.7
30% Server Reduction	63.5	89.0	94.9
% Change	12%	19%	22%

Table III: Lists 50th, 75th, and 95th percentiles of RPS/Server during the two stages of the server pool D reduction experiment. The count of active servers during each stage agreed with the expected 10% reduction, but the RPS/server increased by 22% at each the 95th percentile due to increased traffic load to the service during the experiment.

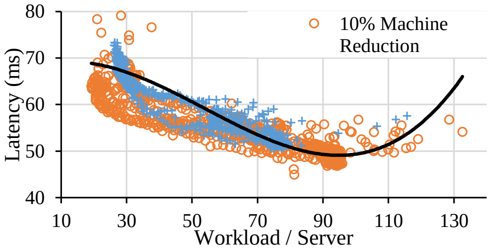

Figure 11: Average 95th percentile latency versus average workload per server for server pool D in datacenter 1 follows our quadratic prediction.

The experiment testing our ability to forecast average CPU and latency under higher load was executed by reducing the server count by 10% for two days. The data collected during this phase of the experiment is shown in Fig. 10 and Fig. 11 as orange data points. As with the service B experiment, reducing server count does not ensure the expected load increase and in this tests, the load distribution shifted to 95 RPS / server a 22% increase.

A linear model trained on the original server pool size

$$y = 0.0916 * RPS + 5.006 \quad R^2 = 0.940 \quad N = 576$$

would have predicted an average 13.7% utilization processor/server at 94.9 RPS/server. Results from the 10% server reduction stage show average processor utilization hit 13.3%

$$y = 0.0892 * RPS + 4.843 \quad R^2 = 0.969 \quad N = 864$$

in line with the forecast value.

Fig. 11 shows the average 95th percentile of internal latency response to load increases. A quadratic polynomial trained on the original server pool size

$$y = 4.66e^{-3} * RPS^2 - 0.80 * RPS + 86.50$$
$$R^2 = 0.90 \quad N = 576$$

forecast an average latency of 52.6ms, while the observed latency at the 95th percentile of load was 50.7.

The same experiment was replicated in data center D4 and produced similar forecast accuracy for both processor and 95th-perc. internal latency. The expected and observed processor were both 15.5 after a 29% increase in RPS/server, while the expected and observed 95th-perc. internal latency shifted from 59 to 61ms.

B. Overall Capacity Saving Opportunity

Now that we have shown our method works for 2 representative micro-services, we analyzed the potentially capacity saving opportunity across the overall large online service running across hundreds of thousands of servers. Since testing our approach at full scale was impractical, as the servers belong to different organizations, etc., we estimate the potential savings by analyzing the server utilization across micro-service pools used by the service. Our intent was to identify a theoretical maximum potential efficiency as an indication of overall system utilization, and an indication of the practical maximum efficiency to tell us how viable it may be to reclaim servers without significant impact to service quality. As others have seen [7], [8], we found a significant portion of server resources are idle as shown in section III-B1.

We applied our methodology to server pools across the large online service we investigated, and show the results for the 7 largest in table IV. It shows a potential savings of 30% of servers while impacting QoS (latency) by an average of 5ms which is less than 1% of the overall service latency, and was considered negligible. 20% of these savings are from running each server pool with less servers while still maintaining an acceptable QoS, and 10% are from improving server availability by changing planned maintenance practices of how many servers are taken offline, and for how long, during software and configuration deployment. Section III-B2 describes our server availability analysis in detail.

1) Savings From Headroom Elimination: We calculate global utilization by taking the sum of the normalized usage over the day. This gives us the theoretical maximum utilization of the current system, indicating the upper bound for efficiency gains. This efficiency is unlikely to be achieved because it assumes we could perfectly partition and mix workloads across servers accurately enough to maintain this throughout the day, and assumes we have a fixed workload is CPU bound. From our analysis of the entire set of servers, we found the global utilization to be 23%. This indicates we have the upper bound for nearly 4x potential for CPU efficiency improvement.

Server Pool	Efficiency Savings	Latency (QoS) Impact	Online Savings	Total Savings
A	15%	9ms	4%	**19%**
B	33%	2ms	27%	**60%**
C	4%	7ms	7%	**11%**
D	33%	8ms	0%	**33%**
E	33%	2ms	2%	**35%**
F	33%	4ms	0%	**33%**
G	5%	1ms	0%	**5%**
***Savings*[5]**	***(20%)***	***(avg. 5ms)***	***(10%)***	***(30%)***

Table IV: Summary of Server Savings for the seven largest server pools, across all datacenters.

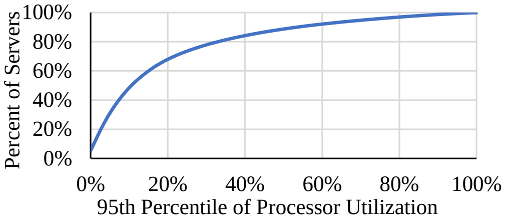

Figure 12: CDF of how many servers are at each percentage of 95th percentile utilization each day.

To understand the theoretical efficiency target we examined the variation of resource utilization for individual servers over the course of the day. We first recorded the 95th percentile value of CPU utilization, which indicates that capacity could be reclaimed without impacting the service. Next we examined the impact of prolonged high server utilization on QoS.

Figure 12 shows the distribution of servers versus their maximum CPU usage for a typical day. We see several interesting things. First, 60% of all servers exhibit a 95^{th} CPU utilization of 15% which is a significant number of underutilized servers. Second, there is a wide variation (between 30% and 100%) of 95^{th} CPU utilization across 20% of the servers. This indicates a relatively small population of servers have high utilization. These results were surprising because we expected at least one CPU intensive event, such as process startup, would cause nearly all machines to have a high 95th utilization value. Service owners told us the over allocation of capacity was to absorb unexpected increases in traffic and unplanned capacity outages.

To understand the duration of servers with high maximum CPU utilization, we examined the distribution of individual 120 second CPU usage samples over the course of the day to understand how they varied. Figure 13 shows it is rare for CPU usage to be above 25% at any point during the day. Only 1% of samples were greater than 25% and fewer than 0.1% of samples were above 40%. This indicates even though 50% of servers have a maximum usage greater than 25% from Figure 12, Figure 13 indicates these are short and rarely occurring spikes, which aligns with our server pool analysis which has no samples above 50% utilization. This shows there are few instances of short duration where high CPU utilization will slow request processing and thus impact latency. Figure 12 tells us fewer than 15% of machines exhibited these spikes of more than 40% utilization. Overall this indicates these servers could use significantly more CPU and still expect few instances of short duration where high utilization impacts latency.

2) Savings From Improving Server Availability: We analyzed the availability of servers to understand how often they are online and measured how much capacity was required to compensate for this unavailability.

We measured the percentage of time each server was online daily, independent of the server pool they were part of, and found the overall average availability was 83%. The distribution of server availability in figure 14 shows most servers are online at least 80% of the time, with a large population at 85% and 98%. Given the large population at 98%, our intuition is that planned maintenance practices could be improved to increase the availability of all servers to this level. Service owners told us the servers online less than 80% of the time were part of server pools re-purposed during non-peak hours to run offline validation of software changes.

Next we investigated the correlation between server availability and server pool availability to know if specific pools were more highly available or if servers with low availability were randomly found in all pools. Figure 15 shows the availability for 3 of the largest server pools C, D, and H, across all data centers. It shows the availability of servers within a pool is quite constant, where pools D and H consistently had 98% availability while pool C consistently had 90% availability, leading us to believe the reasons for unavailability were likely related to server pools as opposed to individual servers. We also see the variation in pool availability is small from day to day, though we do see occasional major unavailability days as we see at the start of the period for pool D.

Planned maintenance of server pools requires the workload to be supported at the required quality level while a subset of servers are taken offline. Service owners told us the primary planned maintenance task was updating software, configuration, and data. The offline event varied in duration depending on the time to drain in-flight requests, apply changes, and restart to receive new requests. The secondary reason was planned infrastructure maintenance for operating system upgrades, network changes, and hardware repairs. We expect availability loss from infrastructure maintenance to be uniform across all servers, so the overhead of this can potentially be estimated as one minus the availability of the most available servers (100% - 98% = 2%).

C. Regression Analysis

We conclude our evaluation by showing an example where our regression analysis enabled finding a bug which was previously deployed to production. We used our offline regression analysis to validate a software change for eliminating a memory leak. The system confirmed the change fixed the memory leak, though found it introduced a new defect causing a significant increase in latency of the server pool under higher workloads. Deep engineering investigation initiated by this result found a complex design flaw, which was then rectified. Fig. 16 shows the results of a software change to eliminate the memory leak and the overall latency regression.

We believe this example shows there is value in analyzing the magnitude of resource and latency impact from changes at scale before deploying them to production. As shown here, this can catch regressions and can serve as a potential measurement of the resource and latency cost of a new change. The ability to measure this cost enables budgeting of resources and latency for individual features, whereas without such a system it is only possible to coarsely identify this cost as part of all features running within a server pool.

IV. Related Work

Cloud system efficiency research has focused on increasing utilization and workload throughput by optimizing VM placement on servers to over provision or collocate independent workloads [3], [8], [44], [45]. Our work optimizes for low overall request processing latency in addition to overall throughput. The service we investigated is composed of a single distributed workflow with hundreds of components that run directly on the server hardware whereas the previous work focused on optimizing a large number of smaller independent workloads.

Previous analysis of an online travel service [46] focused on characterizing user session patterns, described the workload and resource variation over time, and described the relationship between latency and workload increases. This analysis was on a much smaller scale and shorter time period, covering only 35 servers in a single location for seven days.

Improving server utilization by running additional workloads is another related area. These follow a strategy of running additional workloads on the same physical server with lower resource guarantees [10]–[13]. Our work is compatible with this solution, by focusing on reducing minimizing the overall capacity of the system we make the workload response more predictable and therefore easier to manage which minimizes the need to locate additional workloads and maximizes the resource guarantees they can have. In addition, our analysis evaluates the impact of higher

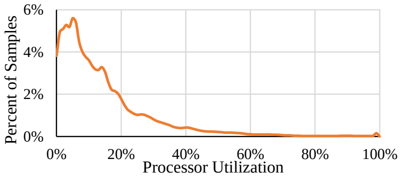

Figure 13: Percentage of samples versus the percent of CPU utilization for all servers over a day.

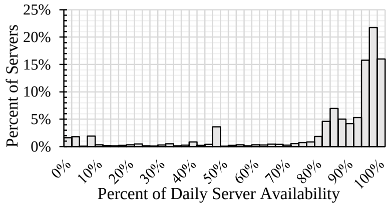

Figure 14: The distribution of servers by percent they are online over a typical day.

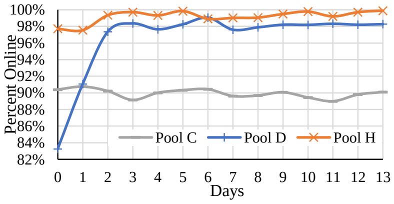

Figure 15: Percentage of daily available pool capacity.

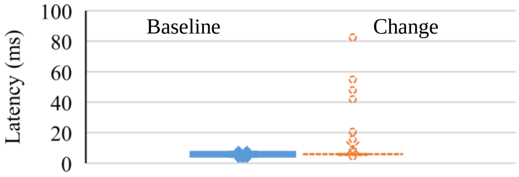

Figure 16: Box plot of latency at each workload interval during a regression test. Each column contains the observed average latency across all servers in the pool. The left results are the baseline, the right are from the change.

server utilization on the QoS (such as end-user latency) of the micro-services running on the servers, illustrating the potential impact of mismanaging secondary workloads.

Another approach to increasing capacity efficiency has been to dynamically add or remove servers as demand changes [2], [3], [47]–[53], or even turn off unused server capacity [54]. In practice this approach is difficult to implement for large online services because it requires existing systems to be ported to run on top of a new infrastructure, and the complexity of dynamic capacity makes the system difficult to debug. Dynamically enabling capacity assumes capacity is available in the specific datacenter topological location when needed, which is inefficient and difficult to guarantee when thousands of servers may be required instantaneously. Our analysis investigates the benefits of moving workload requests closer to the existing capacity because this requires less operational overhead and eliminates the lag time to bring capacity online.

Our work builds on research into the area of tools and strategies for optimizing distributed system software performance [55]. We show our results from comparing new releases of software to their previously released version to detect regressions in scale and performance. Our approach is able to support validating changes that require more capacity than a single server, yet does not require the entire distributed system to validate the change.

Custom hardware has been used to make efficiency improvements [56]. With the transition to cloud computing, service owners will not be able to make hardware choices because cloud providers will achieve greater economies of scale by minimizing variation in the servers it uses. This will require service builders to focus on maximizing the utilization and performance of their software on common cloud hardware, using techniques similar to those provided in this paper. Simply allocating additional headroom is a prohibitively expensive solution, and cloud providers are not incentivized to tell service owners they can run with less.

Research into data center efficiency has focused on optimization of physical resources such as power and cooling both at a macro level of the datacenter itself, and also of the individual servers [57]–[60]. We believe it is more cost- and energy-efficient to have fewer servers overall with consistent usage than to have large numbers of dormant machines. Any improvements in this area are complimentary to efficient server allocation.

V. Conclusion

We presented a summary of server capacity usage across a large online service containing on the order of hundred thousands of servers, shared a novel methodology for capacity planning that forecast significant server savings (20% to 40%) and confirmed these results experimentally. One of the key lessons learned was if you blindly measure the resource usage of your system there will be too much noise in the results to use for capacity planning. However, partitioning the resource usage data by workload enables us to clearly see the linear relationship between workload and resource usage which enables our simplified yet effective black-box capacity planning approach. In addition, without accurate partitioning of micro-service behavior into servers with identical behaviors this approach would not be possible. We found developing metrics for each workload to be relatively simple since workloads are naturally consumed across machine boundaries, however VMs, containers, or process boundaries are sufficient for our approach to work. Given the large number of online services in operation today, the potential savings by applying this work could be significant in terms of servers and money saved as well as the environmental impact of energy savings.

Acknowledgments. This work was partly supported by Microsoft Research through its PhD Scholarship Programme.

References

[1] R. Miller, "Ballmer: Microsoft has 1 million servers," http://bit.ly/16NeFWg, June 2013, [Online; accessed 3-May-2018].

[2] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and QoS-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.

[3] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[4] Google, "Data center locations," http://bit.ly/1prSM6r, October 2013, [Online; accessed 3-May-2018].

[5] Y. Sverdlik, "Google's data center spend slows in 2012," http://bit.ly/1QXIFys, January 2013, [Online; accessed 3-May-2018].

[6] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[7] I. Narayanan, A. Kansal, A. Sivasubramaniam, B. Urgaonkar, and S. Govindan, "Towards a leaner geo-distributed cloud infrastructure," in *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, 2014.

[8] H. Liu, "A measurement study of server utilization in public clouds," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. IEEE, 2011, pp. 435–442.

[9] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 1287–1294.

[10] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes, "Long-term SLOs for reclaimed cloud computing resources," in *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 2014, pp. 1–13.

[11] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. ACM, 2015, pp. 450–462.

[12] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via VM multiplexing," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 11–20.

[13] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*. IEEE, 2007, pp. 171–180.

[14] Y. Zhang, G. Prekas, G. M. Fumarola, M. Fontoura, I. Goiri, and R. Bianchini, "History-based harvesting of spare cycles and storage in large-scale datacenters," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[15] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes, and http chunking in web search," in *Velocity Web Performance and Operations Conference*, 2009.

[16] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.

[17] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using M/G/m/m+r queuing systems," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 5, pp. 936–943, 2012.

[18] J. Hillston, *A compositional approach to performance modelling*. Cambridge University Press, 2005, vol. 12.

[19] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[20] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Injecting realistic burstiness to a traditional client-server benchmark," in *Proceedings of the 6th international conference on Autonomic computing*. ACM, 2009, pp. 149–158.

[21] D. Tang, A. Agarwal, D. O'Brien, and M. Meyer, "Overlapping experiment infrastructure: More, better, faster experimentation," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 17–26.

[22] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at Facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, 2013.

[23] C. Verbowski, E. Kiciman, B. Daniels, Y.-M. Wang, R. Roussev, S. Lu, and J. Lee, "Analyzing persistent state interactions to improve state management," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 34. ACM, 2006, pp. 363–364.

[24] J. Bi, H. Yuan, W. Tan, M. Zhou, Y. Fan, J. Zhang, and J. Li, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Transactions on Automation Science and Engineering*, 2015.

[25] E. F. Coutinho, F. R. de Carvalho Sousa, P. A. L. Rego, D. G. Gomes, and J. N. de Souza, "Elasticity in cloud computing: a survey," *annals of telecommunications-annales des télécommunications*, vol. 70, no. 7-8, pp. 289–309, 2015.

[26] H. Chen, X. Zhu, H. Guo, J. Zhu, X. Qin, and J. Wu, "Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment," *Journal of Systems and Software*, vol. 99, pp. 20–35, 2015.

[27] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu, “Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers,” in *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*. IEEE, 2003, pp. 208–217.

[28] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, “Dynamic cluster reconfiguration for power and performance,” in *Compilers and operating systems for low power*. Springer, 2003, pp. 75–93.

[29] S. Newman, *Building Microservices*. " O’Reilly Media, Inc.", 2015.

[30] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, “Capacity management and demand prediction for next generation data centers,” in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 43–50.

[31] J. R. Hamilton *et al.*, “On designing and deploying internet-scale services.” in *LISA*, vol. 18, 2007, pp. 1–18.

[32] M. Villamizar, O. Garcés, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil, “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” in *Computing Colombian Conference (10CCC), 2015 10th*. IEEE, 2015, pp. 583–590.

[33] S. Tibkin, “Global amazon outage keeps some customers from placing orders,” http://cnet.co/24JFMek, September 2014, [Online; accessed 3-May-2018].

[34] I. Steiner, “ebay down: Twelfth outage of the year,” http://bit.ly/1Nq7NmV, September 2014, [Online; accessed 3-May-2018].

[35] P. Wainewright, “Reputation, trust and salesforce outages,” http://zd.net/1VSFr7u, February 2006, [Online; accessed 3-May-2018].

[36] A. W. Services, “Amazon EC2 Service Level Agreement,” http://aws.amazon.com/ec2-sla/, May 2016, [Online; accessed 3-May-2018].

[37] Akamai, “Terra enterprise solutions: Product brief - global traffic managmeent,” http://akamai.me/1NpMsdc, 2012, [Online; accessed 3-May-2018].

[38] Amazon, “EC2 VM sizes,” http://amzn.to/1SjDBb4, 2017, [Online; accessed 3-May-2018].

[39] T. Mackinnon, S. Freeman, and P. Craig, “Endo-testing: unit testing with mock objects,” *Extreme programming examined*, vol. 1, pp. 287–302, 2001.

[40] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab *et al.*, “Scaling memcache at Facebook,” in *nsdi*, vol. 13, 2013, pp. 385–398.

[41] G. E. Box and K. Wilson, “On the experimental attainment of optimum conditions,” in *Breakthroughs in Statistics*. Springer, 1992, pp. 270–310.

[42] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[43] Microsoft, “Perf_100nsec_timer,” http://bit.ly/1T2Xdma, 2015, [Online; accessed 3-May-2018].

[44] Q. Zhang, J. Hellerstein, R. Boutaba *et al.*, “Characterizing task usage shapes in google’s compute clusters,” in *Large Scale Distributed Systems and Middleware Workshop (LADIS’11)*, 2011.

[45] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, “Towards characterizing cloud backend workloads: insights from google compute clusters,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[46] N. Poggi, D. Carrera, R. Gavalda, J. Torres, and E. Ayguadé, “Characterization of workload and resource consumption for an online travel and booking site,” in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–10.

[47] C. Vazquez, R. Krishnan, and E. John, “Time series forecasting of cloud data center workloads for dynamic resource provisioning,” *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, vol. 6, no. 3, pp. 87–110, 2015.

[48] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame *et al.*, “Optimis: A holistic approach to cloud service provisioning,” *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, 2012.

[49] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, “Large-scale cluster management at Google with Borg,” in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[50] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, and F. Safai, “Self-adaptive SLA-driven capacity management for internet services,” in *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*. IEEE, 2006, pp. 557–568.

[51] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, “Omega: flexible, scalable schedulers for large compute clusters,” in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 351–364.

[52] J. Li, K. Shuang, S. Su, Q. Huang, P. Xu, X. Cheng, and J. Wang, “Reducing operational costs through consolidation with resource prediction in the cloud,” in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 793–798.

[53] A. Uchechukwu, K. Li, and K. Li, “Scalable analytic models for performance efficiency in the cloud,” in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2014, pp. 998–1003.

[54] M. Menarini, F. Seracini, X. Zhang, T. Rosing, and I. Kruger, "Green web services: Improving energy efficiency in data centers via workload predictions," in *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*. IEEE, 2013, pp. 8–15.

[55] L. M. Schnorr, A. Legrand, and J.-M. Vincent, "Detection and analysis of resource usage anomalies in large distributed systems through multi-scale visualization," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 15, pp. 1792–1816, 2012.

[56] M. Awasthi, T. Suri, Z. Guz, A. Shayesteh, M. Ghosh, and V. Balakrishnan, "System-level characterization of datacenter applications," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. ACM, 2015, pp. 27–38.

[57] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33. ACM, 2005, pp. 303–314.

[58] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *ACM SIGOPS Operating Systems Review*, vol. 35. ACM, 2001, pp. 103–116.

[59] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: survey on energy efficiency," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 33, 2015.

[60] B. Kantarci and H. T. Mouftah, "Optimal reconfiguration of the cloud network for maximum energy savings," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE Computer Society, 2012, pp. 835–840.