The Fast and The Frugal: Tail Latency Aware Provisioning for Coping with Load Variations

Adithya Kumar, Iyswarya Narayanan, Timothy Zhu, Anand Sivasubramaniam
Pennsylvania State University

Abstract

Small and medium sized enterprises use the cloud for running online, user-facing, tail latency sensitive applications with well-defined fixed monthly budgets. For these applications, adequate system capacity must be provisioned to extract maximal performance despite the challenges of uncertainties in load and request-sizes. In this paper, we address the problem of capacity provisioning under fixed budget constraints with the goal of minimizing tail latency.

To tackle this problem, we propose building systems using a heterogeneous mix of low latency expensive resources and cheap resources that provide high throughput per dollar. As load changes through the day, we use more faster resources to reduce tail latency during low load periods and more cheaper resources to handle the high load periods. To achieve these tail latency benefits, we introduce novel heterogeneity-aware scheduling and autoscaling algorithms that are designed for minimizing tail latency. Using software prototypes and by running experiments on the public cloud, we show that our approach can outperform existing capacity provisioning systems by reducing the tail latency by as much as 45% under fixed-budget settings.

ACM Reference Format:
Adithya Kumar, Iyswarya Narayanan, Timothy Zhu, Anand Sivasubramaniam. 2020. The Fast and The Frugal: Tail Latency Aware Provisioning for Coping with Load Variations. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3366423.3380117

1 Introduction

Many small and medium sized businesses (SMBs) run online, user-facing, latency sensitive web services such as image recognition [7, 76], voice search [45, 79], etc. These businesses are moving their workloads to the cloud because of the economic benefits realized in terms of reduced capital and operating expenditures [21, 82]. Albeit economically attractive from different perspectives, the pay-as-you-go cloud model can rapidly run up the costs, making it imperative to cap the expenditures over some time horizon while still meeting the desired performance over this period. Further, cloud platforms are also exposing various hardware configurations with different performance price-points [3], making it challenging to apportion the budget over these resources at every instant.

Small and medium sized businesses have two main needs: 1) Constrained by fixed budgets for their IT infrastructure [49], they desire low and *predictable* expenditures, despite *unpredictability in their demand.* 2) As they run user-facing applications, minimizing the tail latency[1] of user requests is critical than optimizing for the average behavior [11, 12]. In this direction, this paper presents a novel approach to using heterogeneity as a capacity provisioning knob for *reducing tail latency* under a *fixed budget constraint.*

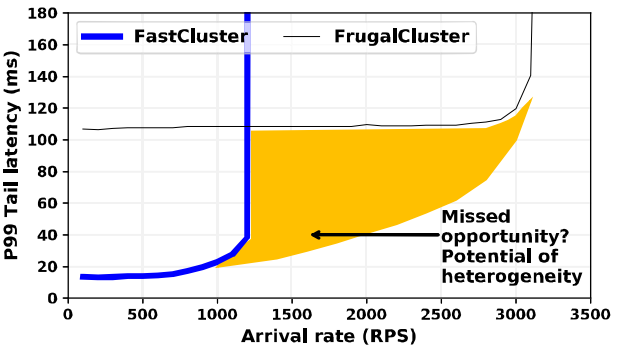


Figure 1: Load in Requests Per Second (RPS) vs. P99 Tail latency of an image recognition web service on a FastCluster and a FrugalCluster for the same cost. Fast server is 5× faster and 15× more expensive than a Frugal server. See Section 6 for details.

Distinction from prior capacity-provisioning and heterogeneity works: The state-of-the-art approach in capacity provisioning is to autoscale the capacity of the cluster dynamically based on load conditions (e.g., [22, 23, 44, 84]). However, these approaches do not satisfy a fixed budget constraint as variability in demand, specifically its "unpredictability" can lead to variability in cost, which is undesirable for businesses that want predictable IT expenditures. In order to ensure a fixed budget constraint, one often resorts to fixed capacity provisioning based on the peak[2] load.

Figure 1 illustrates the trade-off in a fixed budget scenario (iso-cost) where one can afford a few expensive "fast" resources that provide low tail latency or many cheap "frugal" resources that provide high throughput per dollar. A classic example of such resources would be server class Xeon core machines and mobile class Atom/ARM core machines. This trade-off also exists in other scenarios such as a fast (costly) accelerator (GPU/ASICs) vs. a frugal (slow) CPU, and "fast" Non-Volatile Memory (NVMe) storage [48, 90] vs. "frugal" network based storage [27, 77]. Supposing that a workload's arrival rate varies between 1000-3000 requests per second (RPS) (Figure 1), peak provisioning would result in using the slow resources that exhibit poor latency. A better approach, which we use as our baseline for comparison, is to use the fast resources during low load and the frugal resources during high load. We propose to

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380117

[1] In this paper, we use the 99th percentile of latency (P99 Tail Latency) as our tail latency metric.
[2] Provisioning for anything less than the peak load results in periods of poor performance and performance instability, *affecting both the mean and tail latency.*

improve upon this by exploring how to use a heterogeneous mix of such variety of resources to achieve tail latencies in the shaded region of Figure 1.

While there is a large body of prior work [43, 72, 73, 94] that leverage heterogeneous mixes of fast and frugal resources, they have not been studied for user-facing applications where dynamic load variations and tail latencies pose important challenges. Such studies have neither investigated the suitability of heterogeneity to address the variability in load nor do they optimize for tail latency which are two important goals for user-facing applications.

Heterogeneity as a provisioning knob: Our novel idea is to dynamically adjust the mix of fast and frugal server types according to the load while adhering to a desired cost. Fast servers minimize the tail latency and the frugal servers enables us to scale well under load variations. When load increases, we gradually swap a few of the costly fast servers for many frugal but slower servers, and vice versa when load decreases. For instance, if the frugal servers are half the cost of fast servers, we would swap one fast server for two frugal servers. By using heterogeneity as a provisioning knob, we can adapt to load increases by changing the heterogeneity composition rather than incurring additional cost via more servers.

Challenges of heterogeneous clusters: In a heterogeneous cluster of fast and frugal servers, one might expect that the tail latency is somewhere between that of a homogeneous cluster consisting of only-fast (FastCluster) or only-frugal (FrugalCluster) servers. We observe that simply replacing homogeneous servers with a heterogeneous set of servers (i) does not provide the tail latency benefits of a FastCluster and (ii) does not scale as well as a FrugalCluster. We find that this is because even state-of-the-art load balancers and schedulers [4, 70] naïvely send large requests to the slower servers, which significantly impacts tail latency. Tail latency is notoriously difficult to handle, and we demonstrate in this paper that it is critical to design schedulers for both heterogeneity and tail latency to realize the benefits of a heterogeneous cluster. Our work is not just applicable for fast and slow compute servers but goes further to exploit performance trade-offs arising out of heterogeneity in other resources namely, memory capacity and storage systems (Section 7).

Three-pronged solution: First, we introduce our tail latency aware heterogeneous capacity provisioning system for user facing applications that dynamically adjusts the heterogeneity composition of the cluster based on load and a desired cost. Second, we design a new heterogeneity-aware scheduling policy for minimizing tail latency. The policy preferentially schedules requests to faster servers over the slower servers, and when it detects that a faster server is under-utilized, it steals a mismatched request from the slower servers and restarts it on the faster server. Third, we support two modes of allocating the fixed budget across time. While equally dividing a monthly budget across time periods (e.g., an hour) is simplest, we show that allocating cost in proportion to the estimated load can yield more stable performance. This is mainly due to the ability of our system to gracefully adapt to load mispredictions by adjusting the heterogeneity composition dynamically.

Contributions:

- In this work, we make a case for dynamically adjusting the heterogeneity composition of a cluster based on load in order to minimize tail latency. Heterogeneity allows flexibility in the trade-off between low latency clusters and high throughput scale-out clusters. Under a fixed budget constraint, this flexibility is critical for dealing with load fluctuations throughout the day.
- To attain the benefits and flexibility of heterogeneity, we introduce a novel scheduling policy that effectively combines the benefits of low-latency servers and scale-out servers. Our scheduling policy optimizes for tail latency and is robust to the presence of request-size variability.
- We evaluate our solution on real-hardware using an in-house cluster with 75 Atom servers and 5 Xeon servers. Our in-house prototype system performs capacity provisioning for two representative latency sensitive applications (an image recognition deep neural network and the sphinx speech-to-text application) with diurnal load variations. Our evaluation on these two applications shows that we can reduce the tail latency of the applications by 42% - 45% on average using our heterogeneous design.
- We also evaluate a database application in the public cloud and show how the "fast" and "frugal" trade-off can apply to memory capacity as well as storage systems.
- The broader implication of our study is that heterogeneity can be a useful knob to even mitigate the impact of mispredictions of load, which is a common problem in capacity provisioning for high variability workloads.

2 Background

2.1 Hardware diversity on the cloud

Heterogeneity exists in plethora of forms on the cloud. In this work, we consider heterogeneity arising from hardware configurations with cost and performance differences. For instance, high performance Xeon-servers can deliver 10× more performance than Atom-servers, but are $50-100\times$ more expensive. Cloud vendors have begun to offer ARM core machines [5], Raspberry-Pi clusters [6, 63], and Atom core machines [19]. Beyond compute, heterogeneity exists in storage tiers as well. Solid state disks (SSDs) and the recently introduced Non-volatile memory (NVMe) disks are extremely fast and expensive while traditional disk-based and network based storage are cheap and slow. Very recently in the cloud space, providers like AWS have introduced schemes [47] with flexibility in choosing instance types with a commitment towards usage (in $\$/hr$).

2.2 Workload variability

A user-facing application's load often varies over time. For example, Figure 2 shows the load on Wikipedia for a typical one week period. As seen, there is diurnality in the load through the day and week resulting in 2-8× gap between the peak and low load. It is inherently difficult to accurately predict the application load even using sophisticated prediction mechanisms [17]. For example, we analyze the load unpredictability of Wikipedia by comparing the relative difference in load for each time slot across the three weeks following the trace shown in Figure 2. We compare the corresponding day-of-the-week and hour-of-the-day behavior and find that the load can vary by 1%-10% of the peak compared to the Figure 2 load in the corresponding time. Workload variations and uncertainties pose a critical challenge while provisioning as resource deficiency causes performance violations of the application especially the tail. This is further complicated by request level variations in the amount of work to be done (referred to as job size). This work proposes to

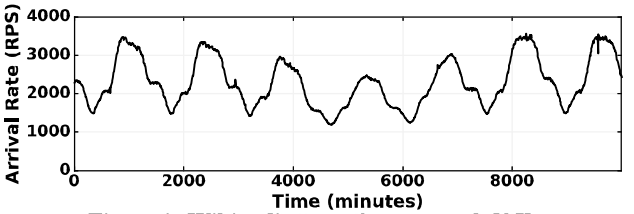


Figure 2: Wikipedia trace for one week [85].

use heterogeneity as a knob to cope with the variability and unpredictability in the workload.

3 Motivation

Small and medium sized businesses require a strict control over their IT expenditures for survival [71]. Under this constraint, there are two approaches to handle the key challenge of load variation and uncertainty in capacity provisioning. One could equally subdivide the budget across time (e.g., $\$3,000/mo \rightarrow \approx \$4/hr$). In homogeneous clusters, this translates to fixed capacity provisioning and under-utilization during low loads. Alternatively, one could subdivide the budget based on estimated load within the budget planning horizon (e.g., month or week). This is the autoscaling analogy in fixed budget capacity provisioning, but mispredictions could result in insufficient capacity and poor performance. Instead of trying to predict load, this paper shows how the flexibility of dynamically adjusting heterogeneity compositions can be used to solve the problems in both of the approaches for handling load variation under a fixed budget.

3.1 Heterogeneity to address provisioning challenges

We focus on hardware heterogeneity where a fast resource type provides low latency albeit at a high expense whereas a cost optimized resource type provides high throughput per dollar albeit at higher latency. While much of the prior effort in tackling capacity provisioning challenges (e.g., [24, 59, 74]) use homogeneous clusters, we explore how leveraging a heterogeneous mix of resources can improve tail latency in fixed cost budget constrained environments.

Figure 3 illustrates where heterogeneity can help in handling load variation. Figure 3 (a) shows a snippet of the Figure 2 Wikipedia trace (scaled, See Section 6) and Figure 3 (b) adds a heterogeneity configuration to Figure 1. At low load regions (marked as ② in figure 3), a few fast servers can handle the entire load and provide low tail latency because of their superior processing speed. This is much better than a "FrugalCluster" configuration with slower processing speeds. That is, at low load regions, it is better to *scale-up* within a server to improve service times than to *scale-out* the number of servers. However, at high load regions (marked as ① in figure 3), it is too expensive to provision only fast servers, so *scaling out* with slower servers is necessary to adhere to low fixed budgets.

The above observation may suggest that an "FrugalCluster" or "FastCluster" option may be ideal, as long as we are allowed to switch between these two options based on the load. We refer to this as *temporal heterogeneity* where the cluster is always homogeneous, but it can change between all-fast and all-slow configurations over time. While prior works have exploited temporal heterogeneity mainly to optimize for energy during different load periods [36, 54, 57, 75], temporal heterogeneity does not achieve the desired performance in the region highlighted in Figure 1. This region corresponds to a medium load region (e.g., ③ in Figure 3 (a)), which often constitutes a large fraction of time (e.g., 49% of the time in the Wikipedia trace is between 1500 and 2800 RPS). In this paper, we show that *spatial heterogeneity* composed of a mix of fast and frugal servers, can help in reducing tail latency in this region as illustrated by the Heterogeneous line in Figure 3 (b). When the budget is equally subdivided across time, our system dynamically switches between different spatial heterogeneity (henceforth referred to as heterogeneity) compositions based on the measured load. That is, as load increases, we swap out fast servers with multiple frugal servers and vice versa as load decreases. The flexibility of gradually transitioning between heterogeneity compositions allows our system to scale to the current load while maintaining a fixed cost, albeit with varying performance.

When the budget is subdivided based on estimated load, the flexibility of changing heterogeneity compositions is also helpful in handling load mispredictions. If load is higher than expected, then our system will transition to a heterogeneous composition with slightly more frugal servers so as to handle the extra load. By contrast, a system that only supports temporal heterogeneity is often stuck with an all-slow configuration when there are mispredictions.

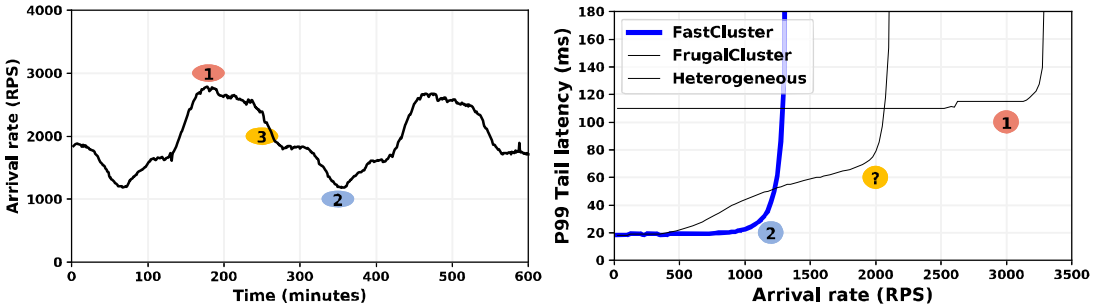
3.2 Challenges in scheduling with heterogeneity

The benefits of spatial heterogeneity over temporal heterogeneity rely upon the assumption that a heterogeneous mix of servers yields better performance than an all-fast or all-slow cluster at the same cost. We find that this is not always true, and a critical factor for providing low tail latency in a heterogeneous cluster is the scheduling policy. If one extends the random load balancing policy to be heterogeneity-aware by assigning different weights for fast and slow servers, it surprisingly does not provide any performance benefit for any iso-cost heterogeneous mixture. Figure 4 shows the arrival rate vs mean latency for different heterogeneous mixtures of fast and slow servers. Even for mean latency, there is no benefit; tail latency results (not shown) are similar. Even though the weighted load balancing policy is heterogeneity-aware and balances the load according to the speed of each server, this is not sufficient for optimizing latency for either the mean or tail. As we will explore in depth later in Section 4.1, our more sophisticated scheduling policies are needed in heterogeneous configurations for reducing latency, especially for tail latency.

4 Navigating the design space with queueing theory

In designing our solution, we decompose our problem into three key pieces. First, we need to design a heterogeneity-aware scheduling policy that avoids getting large requests stuck at slow servers, which widens the tail latency. Second, we need to decide when to switch to different heterogeneity compositions based on load and budget. Third, we need to choose a policy for subdividing an overall (e.g., weekly) budget across time slots (e.g., minutes). Towards this, we explore solutions for this coupled provisioning-scheduling problem with fixed cost budgets by using queueing theory to derive insightful heuristics. Using these insights, we build and evaluate an actual prototype leveraging heterogeneity (Section 5).

Queuing simulation setup: A queueing simulator allows us to study interactions between different aspects of the application behavior (load variations, request size variations), hardware properties (heterogeneous cluster size, service rates), and the system policies (scheduling, resource allocation); a very wide design space that is not easily



(a) Wikipedia trace showing diurnal arrival rate pattern (b) Relationship between arrival rate and tail latency (iso-cost)

Figure 3: Motivation for heterogeneity under varying loads.

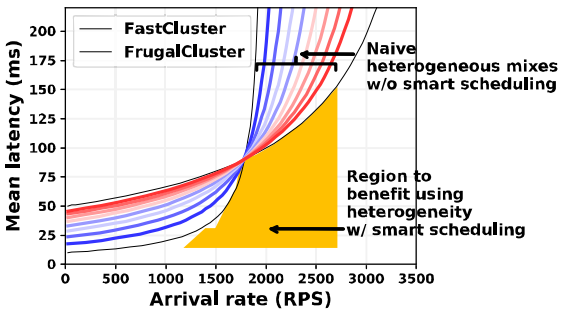


Figure 4: Analytical calculation by taking a weighted probability on the closed form expression for mean latency of individual M/M/1 queues. Fast servers are 5× faster than slow servers. Highlights the need for better scheduling for effective utilization of heterogeneous resources.

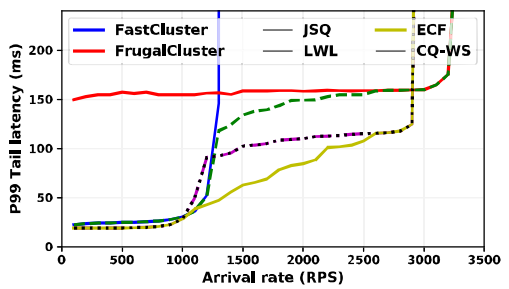


Figure 5: Comparing the P99 tail latency for different scheduling policies with homogeneous and heterogeneous mixes with the same cost. A fast server costs 15× a frugal server.

feasible to study on an actual platform. We methodically analyze this design space using an in-house simulator parameterized as follows. The cluster is modeled as FIFO queues with N_i servers each with service rates μ_i (i: 1 → n). When a request arrives, a load balancer assigns it to one server based on the scheduling policy. For the policy with a central queue, there is a single central FIFO queue and all servers get work assigned from that central queue. Request arrivals follow a Poisson process with exponential request size distributions[3]. Experiments are repeated 100 times with different random seeds and the mean values are plotted.

Metric of interest: We use the metric of iso-cost performance (as in Figure 3(b)) to study the trade-offs between different heterogeneous cluster compositions of the same cost. That is, we measure the 99th percentile tail latency across different heterogeneity compositions with the same cost across the load spectrum. We study these curves and propose heuristics to plan, allocate, and operate resources, which we implement and evaluate in Sections 5 and 6.

4.1 Scheduling

Given a mix of heterogeneous servers, how do we extract the maximum performance? As shown in Section 3.2, naïve load-balancing strategies fail to extract good performance, so we next explore different scheduling policies for assigning requests to servers.

Join-the-Shortest-Queue (JSQ): This policy assigns requests to the server with the shortest queue length (e.g. Nginx [70]). Figure 5 shows that JSQ is better than weighted random load balancing and provides some benefits under heterogeneous provisions but is not ideal as it treats all requests alike i.e., it does not know request sizes.

Least-Work-Left (LWL): Unlike JSQ, LWL sends the request to the queue which has the minimum amount of work left to do. Apart from the queue lengths, the scheduler also needs to know the remaining request sizes at each queue. As seen in Figure 5, LWL offers better performance than JSQ. However, it is oblivious of server heterogeneity while assigning an incoming request to a server.

Earliest-Completion-First (ECF): We propose extending LWL to be heterogeneity aware by biasing requests towards fast servers when available. Our scheduler, Earliest-Completion-First, selects the server where the request would be completed earliest. With homogeneous servers, ECF is equivalent to LWL. With heterogeneous servers, the fast servers will be used first until there is sufficient queueing at all the fast servers where running the request on a frugal server is faster. As seen in Figure 5, ECF, which accounts for both heterogeneity and request size variability, provides better performance at the tail compared to all other policies.

Central Queue + Work-stealing (CQ-WS): Both LWL and ECF require explicit knowledge of request sizes, which may not be available in practice [67]. Under such conditions, the scheduling policy should dynamically identify and adapt to request size variability while accounting for the different processing speeds of the servers. To handle request size variability, we utilize an idea from queueing theory: systems that use a common central queue are equivalent to systems that immediately dispatch requests to servers based on the request size aware LWL policy [39]. Hence, we propose to enhance the central queue approach to be heterogeneity aware by (i) biasing requests towards the fast servers when available, and (ii) having the fast servers steal work from the slower frugal servers when idle.

[3] Results from more varying hyper-exponential request size distributions [58] with larger squared coefficient of variation (CV^2 value >> 1) follow similar trends.

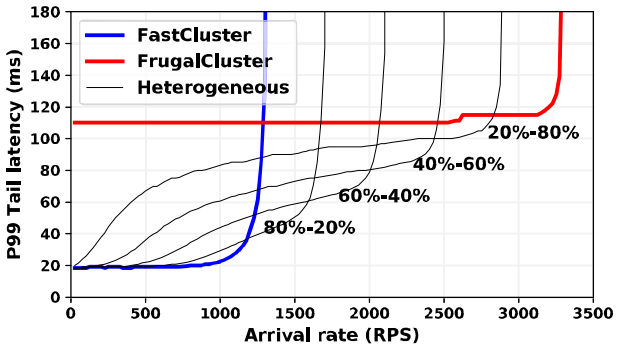


Figure 6: Graphical representation of iso-cost curves. Heterogeneous mixes are denoted by fraction of budget allocated to fast/frugal servers as Fast%-Frugal%.

Our CQ-WS scheduling policy works as follows. On request arrival, the request gets assigned to a fast server if there is one free. If all fast servers are busy, it gets assigned to a free frugal server if available. If all servers are busy, it joins the central queue. Whenever any server becomes free, it pulls a request from this central queue. To mitigate the pitfalls of having a large request assigned to a slow frugal server in a request size oblivious manner, we have the fast servers perform work stealing (i.e., the request is canceled on the frugal server and restarted from the beginning on the fast server). Through experimenting with many work stealing policies,[4] we find that a good and simple policy is to have a fast server steal from a random frugal server when it finishes a request and finds an empty central queue.

Key Takeaway: A good scheduler needs to account for both request size as well as service time variability in a heterogeneous cluster. If request sizes are known, ECF is a good scheduling policy, and when unknown, the CQ-WS policy does quite well.

4.2 Dynamically adjusting heterogeneity

Assuming that a heterogeneity-aware scheduling policy is chosen, we next consider how to select the best heterogeneous mix of servers given a fixed budget for a time slot. As the answer depends on the arrival rate of requests, we first generate offline graphs of the system performance at various arrival rates and heterogeneity compositions. We call the results "*iso-cost curves*", which capture the following information: (Cost, Arrival rate, % Composition of (fast/frugal) servers, Tail latency). We narrow the data based on the given budget and then identify the heterogeneity composition with the lowest tail latency for each arrival rate. Figure 6 shows a visual representation of the iso-cost curve. Using the iso-cost curve for the given budget, we measure the current arrival rate and select the heterogeneity mix with the lowest tail latency as indicated by the iso-cost curve. As the change in heterogeneity composition between time slots is gradual, there is a limited overhead in switching the servers, which we find to be small (see Section 6.1) in our system evaluation. For systems that exhibit larger overheads, it is possible to preemptively start the transition beforehand, though we leave this to future work. Similarly, we leave more frequent adjusting of heterogeneity to future work.

[4]Variants include hoarding a pool of requests exclusively for the fast servers and stealing the longest/shortest running request amongst frugal servers.

Key Takeaway: Using offline precomputed data about system performance for each heterogeneity composition, we select the best configuration for the allocated budget at the measured arrival rate.

4.3 Budget allocation across time slots

Given an overall (e.g., monthly, weekly) budget, we consider two policies for dividing the budget across time slots (e.g., hours, minutes): equal allocation and load proportional allocation.

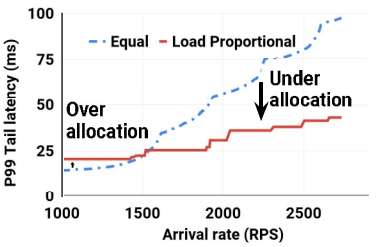


Figure 7: Better balance of latency with load proportional allocation

Equal allocation: With no a priori knowledge of the load, the simplest policy is to equally apportion the available budget across all time slots. Under such a policy, our approach of dynamically adjusting heterogeneity will allocate more frugal servers during high load and more fast servers during low load. This will result in a performance that varies with load. By contrast, a temporal heterogeneity solution will have poor performance in all but the low load periods.

Load proportional allocation: To compensate for load variability, we propose a load-proportional cost allocation policy where each time slot gets a fraction of the total budget proportional to its estimated load. Figure 7 compares this policy with the equal allocation policy wherein we observe a higher tail latency with equal allocation policy. Unlike the equal allocation policy that is constrained to spend during high loads, the load proportional policy saves spending during low load periods in order to obtain a more balanced latency across time.

Key Takeaway: If load estimates are available, load proportional budget allocation is better than equal allocation for lowering performance variability over time assuming we dynamically control the heterogeneity composition and use a good scheduling policy.

5 From insights to system prototype

Using the insights from Section 4, we implement a system prototype and deploy it on a heterogeneous cluster in our laboratory comprising Intel Xeon E5 2680 servers and Intel Atom N570 servers as well as the cloud. Figure 8 shows the three components in our implementation: the Offline profiler, the Online resource provisioner & scheduler, and the Online load estimator & budget allocator.

Offline profiler: The offline profiler is responsible for measuring the application performance on the heterogeneous resources and generating iso-cost curves described in Section 4.2. It runs the application on each server type (e.g., Xeon server and Atom server) and obtains the maximal application service rates that can be processed by one server for each type. To accelerate the generation of the iso-cost curves, we resort to simulation using the measured service rates. As identifying the best performing heterogeneity composition for a given load is a relatively simple task (as compared to performance prediction), we find that simulation works reasonably well. We use the simulator to sweep over a range of load values between the historical minimum and maximum load values of this application for each heterogeneity composition to generate the iso-cost curves.

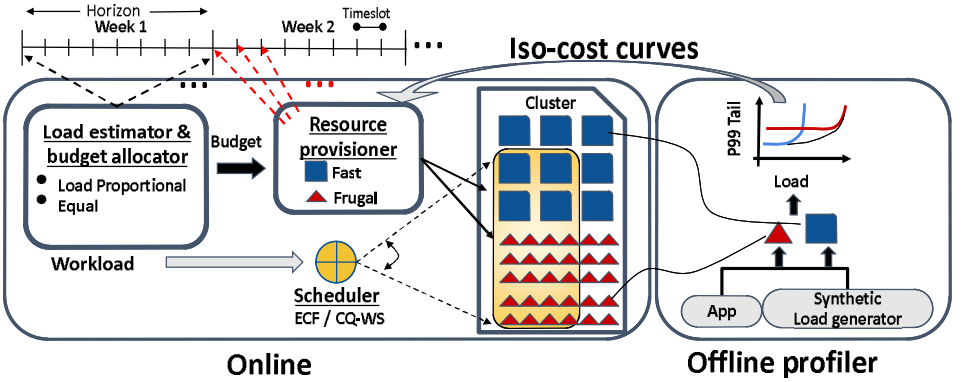


Figure 8: Our implementation consisting of 3 components: (i) Offline profiler, (ii) Online resource provisioner & scheduler, (iii) Online load estimator & budget allocator.

Online resource provisioner & scheduler: The online resource provisioner is responsible for selecting the heterogeneity composition for the current time slot. At the beginning of each time slot, this module uses the iso-cost curves for the time slot's budget to determine the best mix of servers to provision for the current load. The module acquires new servers as needed in the desired heterogeneity composition, and it then warms up the newly allocated servers.

The list of servers, along with whether the server is fast or frugal, is updated at the scheduler. The scheduler is responsible for load balancing requests among the servers in the cluster. Our scheduler is currently implemented on the gRPC [30] framework for the `img-dnn` and `sphinx` applications (deployed as an RPC service) that we use in our evaluations. Our scheduler can be easily integrated to work with any application deployed using gRPC. We implement the ECF and CQ-WS scheduling policies (and the other scheduling policies for comparison). We use ECF when the application is able to estimate request sizes accurately. Otherwise, it defaults to a CQ-WS scheduling policy, where it keeps track of the idle/busy state of every server. The scheduler leverages request tracking (done in existing production systems [70]) to keep track of the system's state. Based on the scheduling decision, the request is forwarded to the appropriate server by making an RPC call. On average this hand-off incurs an overhead of <100 μs per request and the overall framework is lightweight and implemented in 2 KLOC in C++.

Online load estimator & budget allocator: The online load estimator and budget allocator is responsible for estimating load within a planning horizon and dividing the budget across time slots within the planning horizon. In our implementation, we break the entire execution duration into planning horizons of one week (as shown in Figure 8), but the ideas extend to other horizons (e.g., month, day). At the beginning of each horizon, we perform a load prediction for this horizon at the granularity of 5 minutes (time slots) using the day-of-week-plus-time-of-day prediction [52]. During bootstrapping where load predictions are unavailable, we simply use a constant load predictor (i.e. equal cost allocation). This load estimation is used to split the total budget for the horizon into a budget for each time slot of 5 minutes.

6 Evaluation using real hardware

6.1 Experimental setup

Cluster: We evaluate our prototype both on in-house clusters and public cloud offerings. Our in-house hardware cluster comprises 5 Xeon [51] E5-2680 cores @ 2.5GHz (fast) each costing 15 units and 75 Atom [50] N570 cores @ 1.66GHz (frugal) each costing 1 unit. All machines are bare-metal running the Linux 2.6.32 kernel with gigabit ethernet networking. Descriptions for experiments on the public cloud are deferred to Section 7.

Details	`img-dnn`(Image)	`sphinx`(Speech)
Service rate (RPS)	Fast: 220 Frugal: 40	Fast: 1 Frugal: 0.25
Avg. service time (ms)	Fast: 4.55 Frugal: 25	Fast: 1000 Frugal: 4000
Memory (MB)	400	120
Startup latency (ms)	1105	706
Peak rate used in Wiki trace (RPS)	3000	20

Table 1: Details of the applications used for evaluation.

Application: We examine two applications: (i) `img-dnn` – an OpenCV based image recognition application that classifies input images using a deep neural network, and (ii) `sphinx` – a speech processing application based on the sphinx library from CMU [55]. Both are taken from the tailbench suite [53] and modified to work in our prototype built with the gRPC framework. The details of the application, their memory requirements, service times, and startup overheads are listed in Table 1. On each server, they run as a single threaded application launched with real-time priority with requests serviced in FIFO order. An RPC call takes a set of images or an audio file as input and returns the image classifications or the transcribed text for `img-dnn` and `sphinx` respectively.

Workload: We use the Wikipedia trace as a proxy workload for a web facing application shown in Figure 2 with peak arrival rates scaled for our cluster (see Table 1). The trace provides diurnal traffic patterns that are common in online, user-facing workloads. We generate the load according to a time-varying Poisson process. For `img-dnn`, we incorporate request size variability by coalescing different numbers of images into one request. Since user input size is not predictable for a speech application, we assume no knowledge of request size for `sphinx`. Although we only have space to show results for exponentially distributed request sizes in this paper, we have experimented with other distributions with higher coefficients of variation, and they show that our ideas are still applicable.

Application setup & overheads: There are three sources of system overheads that occur every time slot (5-minutes) when the heterogeneity composition potentially changes: (i) iso-cost curve lookup to determine the heterogeneous cluster composition, which takes $< 200\mu s$, (ii) application startup latency, and (iii) container startup latency (if used). Figure 9 shows the break down of overheads, which are less than 1% of a time slot.

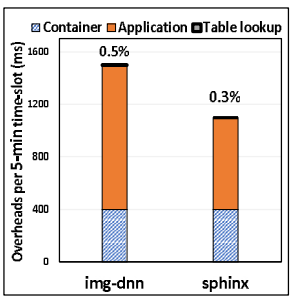


Figure 9: Overheads (<1%).

Cost & budget Settings: The cost of one fast server and one frugal server is fixed at 15 units and 1 unit per time slot respectively matching the price and performance ratios we see in practice. Note that having a budget that is too loose or too strict will result in choosing homogeneous compositions as the only reasonable solution. Consequently, we set a cost budget at 85% of a FastCluster autoscaling solution, which amounts to 222,000 and 335,070 units for `img-dnn` and `sphinx` respectively.

Baseline policy for resource provisioning In selecting a baseline strategy for comparison, we consider state-of-the-art solutions for scheduling and autoscaling while enforcing a fixed cost budget constraint. We find that under existing known scheduling policies, the FastCluster and FrugalCluster configurations provide better tail latency than any mix of fast and frugal servers. Therefore, we consider an autoscaling baseline that takes advantage of temporal heterogeneity to automatically switch between a FastCluster and a FrugalCluster. Based on load, our baseline selects the best server type (fast/frugal) and elastically scales the number of servers [22, 23]. In addition, the baseline's scheduling policy (LWL) is selected to optimize tail latency under homogeneous systems [42]. We have experimented with other alternative baselines such as sending (short/long) requests [18, 40, 41] to (frugal/fast) servers respectively. This is not simple in practice as it requires setting an empirically determined threshold for short/long requests, and this parameter is highly sensitive to the arrival rate and server load conditions.

Outline: Our goal is to show that dynamically controlling heterogeneity compositions can provide low and predictable performance under fixed cost budget constraints despite uncertainties in application behavior. We start by showing how our approach can adapt to load variations (Section 6.2). We then conduct a sensitivity study on how adjusting heterogeneity compositions can cope with load misprediction (Section 6.3). We next evaluate the effectiveness of our CQ-WS scheduling policy for dealing with the lack of request size knowledge, as is often the case in real-world applications such as speech processing (Section 6.4). These aforementioned experiments demonstrate how our solution applies to compute heterogeneity, and we show our approach scales to a moderately sized cluster (~75 machines). Lastly, we consider other forms of heterogeneity in terms of memory capacity and storage systems, which we evaluate in the public cloud (Section 7).

6.2 Adaptability to load variation

We start by studying the ability of heterogeneity to deal with load variability over time while assuming that there is perfect knowledge of request sizes (this will be relaxed in Section 6.4). We study the performance for the following policies:

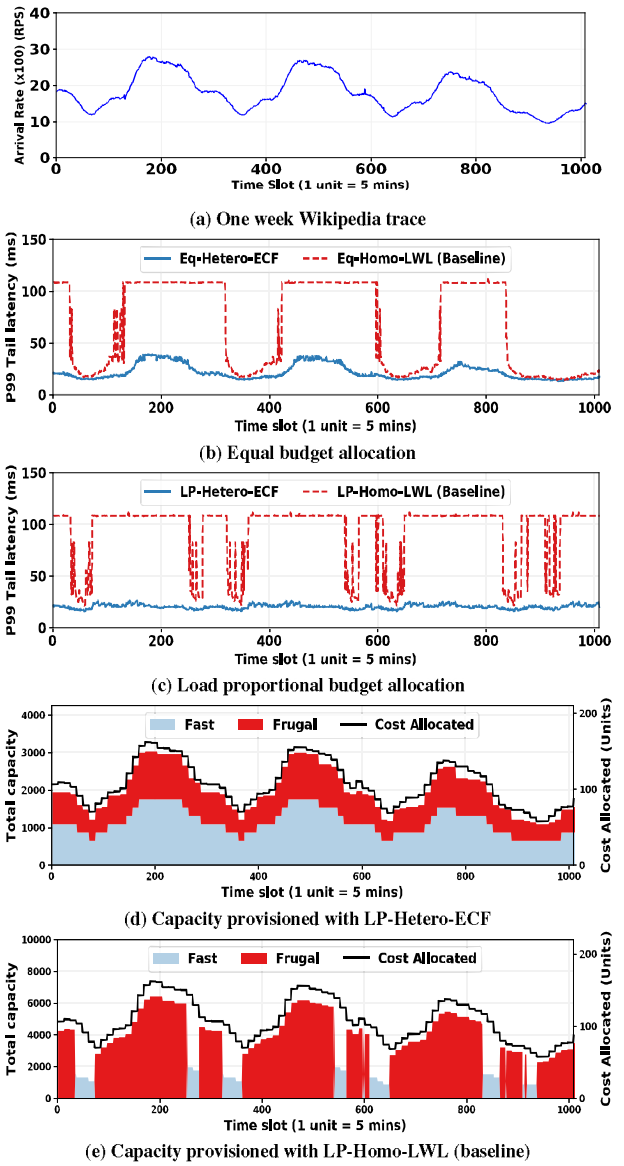
- **Eq-Homo-LWL (baseline):** This policy apportions the cost budget evenly across all time slots. It switches between the homogeneous FastCluster and FrugalCluster provisioning options based on load. It uses the request size aware LWL scheduling policy, which works well for homogeneous clusters.
- **Eq-Hetero-ECF:** Here again, the budget is equally apportioned across all time slots. However, the policy can choose to provision a heterogeneous mix of fast and frugal servers when needed, and it uses the request size aware ECF scheduling policy.
- **LP-Homo-LWL (baseline):** This policy is similar to Eq-Homo-LWL, but performs load proportional budget allocation based on the estimated load. As mentioned earlier, load estimation is done using day-of-week-and-time-of-day prediction.
- **LP-Hetero-ECF:** This policy is like Eq-Hetero-ECF, but divides the cost budget in proportion to the estimated load.

Figures 10b and 10c capture the 99th percentile tail latency (y-axis) of the `img-dnn` application for a period of one week (x-axis) when replaying the Wikipedia load trace (Figure 10a). We show the results from the first 3 days of the trace as rest are similar. Overall, we observe that irrespective of budget allocation policies, heterogeneous cluster mixes help to achieve better tail latency for the entire horizon of one week. Heterogeneity reduces the tail latency by 26% - 83% when compared to homogeneous provisioning strategies across all time slots. *This illustrates the importance of exploiting heterogeneity in fixed budget scenarios, and its benefits in delivering consistently good tail latency with incomplete knowledge of future load.*

Looking at Figure 10b in more detail, we see that equal budget allocation, which does not require a priori load knowledge, results in tail latency that varies with load. Using heterogeneous mixes of fast and frugal servers (Eq-Hetero-ECF) allows for lower tail latency that gradually increases and decreases as compared to switching between homogeneous FastCluster and FrugalCluster configurations (Eq-Homo-LWL) where tail latency oscillates significantly. The standard deviation of tail latencies across different time slots are significant in the homogeneous cluster (σ=42.25 ms) compared to the heterogeneous cluster (σ=7.62 ms). This shows that by exploiting heterogeneity, we can achieve low and less-varying performance (compared to a homogeneous design) even under a naïve budget allocation policy.

Heterogeneity offers much better performance when operating with estimates of load variations. Figure 10c shows that under the load proportional budget allocation policy, the heterogeneous cluster achieves 76% lower latency compared to the homogeneous cluster across all time slots. Moreover, the standard deviation of tail latencies across the slots for the heterogeneous cluster is only 2.04 ms whereas under LP-Homo-LWL it is 29.17 ms (10 reduction) and under Eq-Hetero-LWL it is 7.62 ms (3× reduction). This shows that dynamically adjusting heterogeneity compositions combined with a carefully designed scheduling policy and budget allocation policy can help achieve the key goals of low and consistent performance across the entire planning horizon despite load variations.

Figures 10d and 10e give insight into where heterogeneity is helpful. Under the LP-Hetero-ECF policy (Figure 10d), fast servers help provide low tail latency and frugal servers help scale-out to



(a) One week Wikipedia trace

(b) Equal budget allocation

(c) Load proportional budget allocation

(d) Capacity provisioned with LP-Hetero-ECF

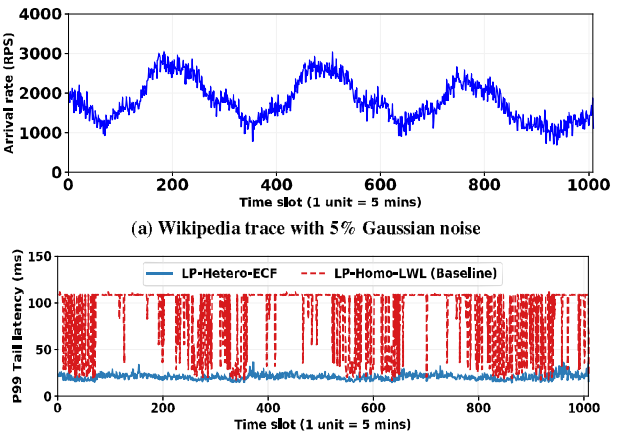
(e) Capacity provisioned with LP-Homo-LWL (baseline)

Figure 10: Adaptability of a heterogeneous design to load variations. Application: `img-dnn`.

the load. By contrast, the LP-Homo-LWL policy (Figure 10e) can use fast servers only during periods of very low loads, fluctuating between FastCluster (low latency) and FrugalCluster performance (high latency).

6.3 Coping with load misprediction

In the prior experiment, we either did not predict the load, or used a fairly accurate day-of-the-week-and-time-of-day predictor. To study the ability to deal with prediction inaccuracies, we next compare the LP-Homo-ECF and LP-Hetero-ECF schemes under load misprediction. Note that the load proportional budget allocation employs the load estimates to divide the budget across time slots. However at runtime, the application behavior could deviate from the estimated load behavior. We perturb the input load by adding a Gaussian noise with a mean of 5% of the peak load (which amounts to a μ=150) for every time slot while replaying the trace. Figure 11



(a) Wikipedia trace with 5% Gaussian noise

(b) Load proportional budget allocation with mispredictions

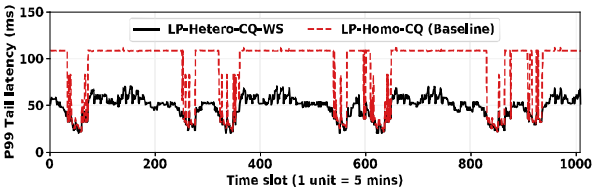
Figure 11: Heterogeneity copes well under load misprediction. Application: `img-dnn`.

shows the performance of both the homogeneous and heterogeneous provisioning strategies for `img-dnn` in the presence of load mispredictions. As can be seen, both strategies incur an increase in the number of time slots with higher tail-latencies compared to operating under perfect load estimations seen previously in Figure 10c. However, the contrast is much more stark for the homogeneous cluster compared to its heterogeneous counterpart. With mispredictions, LP-Homo-LWL is degraded by 16% on average compared to LP-Hetero-ECF, which is degraded by 10% on average. Even when load is mispredicted, LP-Hetero-ECF has multiple heterogeneous options for satisfying the measured load. By contrast, LP-Homo-LWL only has the FastCluster or FrugalCluster configurations to choose from. *This demonstrates that the flexibility of dynamically choosing from a range of heterogeneous configurations can compensate for load mispredictions.*

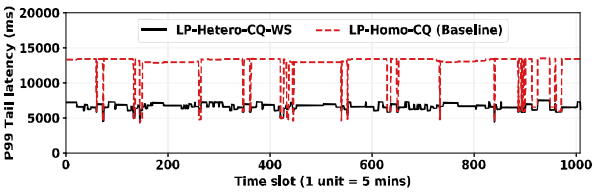
6.4 Coping with lack of request size knowledge

Until now, we have assumed the size of each request is known to the scheduler (ECF), but there exists classes of applications (e.g., speech recognition) where it is impractical to estimate request sizes. For some applications, it is difficult to estimate the size of each request, even when it arrives [67]. To study these, we compare the following schemes, which have no knowledge of request sizes:

- **LP-Homo-CQ (baseline):** This scheme employs load proportional budget allocation and switches between the homogeneous FastCluster and FrugalCluster provisioning options based on load similar to LP-Homo-LWL. The difference is it uses the central queueing (CQ) policy, which does not require request size knowledge, but is equivalent to the request size aware LWL scheduling policy, which works well for homogeneous clusters.
- **LP-Hetero-CQ-WS:** This scheme divides the budget proportionally based on estimated load and provisions a heterogeneous mix of fast and frugal servers similar to LP-Hetero-ECF. The difference is it uses the central queue and work-stealing scheduling policy described in Section 4.1, which does not require request size knowledge.



(a) img-dnn (CQ-WS)



(b) sphinx (CQ-WS)

Figure 12: P99 Tail Latency for the Wikipedia trace under our CQ-WS policy. Heterogeneous provisioning can provide a lower and more consistent tail latency than homogeneous provisioning, even without request size knowledge.

In this experiment, we consider both the `sphinx` and `img-dnn` applications with no knowledge of request sizes. As Figures 12a and 12b show, the ability to adjust heterogeneity (LP-Hetero-CQ-WS) still performs much better (42% and 45% on average across all time slots for `img-dnn` and `sphinx` respectively) than switching between FastCluster and FrugalCluster (LP-Homo-CQ). LP-Hetero-CQ-WS achieves this by preferentially using fast servers initially, and using the frugal servers only upon exhausting the fast servers. Subsequently, when fast servers become free, they can steal (potentially long) requests from frugal servers.

6.5 Is heterogeneity always effective?

The benefit of heterogeneity depends on the overall cost budget. We examine the time slots in the planning horizon that can benefit from heterogeneous design (LP-Hetero-ECF) under different fractions of overall cost budgets for the `img-dnn` application. The magnitude of tail latency improvement is discretized into 3 bins, 1% -30% gain, between 30%-60% gain, and > 60% gain, marked by 3 separate regions in Figure 13. In region 1 with a stringent budget no scheme is adequate. At the other extreme in region 3, a clearly excessive budget would enable the homogeneous FastCluster to perform the best. However, the intermediate region 2 clearly shows the benefit of heterogeneity over homogeneity for a fairly broad budget spread. On the left of this region, we begin to see the benefit of heterogeneity, with much of the latency gains being under 30%. The benefits start amplifying as we move to the right, with 30-60% gains, and even some time slots seeing more than a 60% improvement in tail latency. As we move further, the budget becomes high enough to neutralize the benefit of heterogeneity as FastCluster becomes the ideal solution when one can afford them.

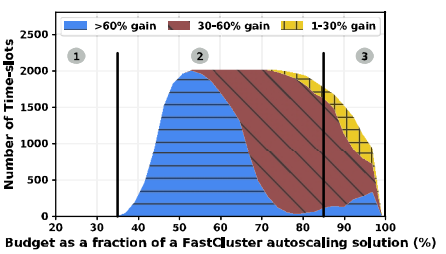


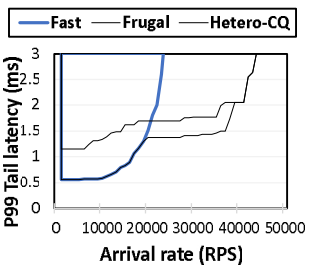
Figure 13: Visualizing the effectiveness of heterogeneity across different cost budgets

7 Heterogeneity of other resource types on the public cloud

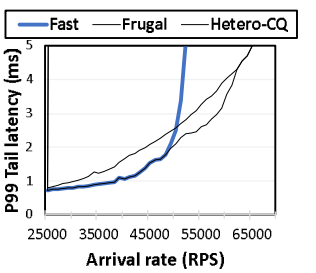
While we have reaped the benefits of heterogeneity for compute intensive applications, data intensive applications are dependent on other resource types such as memory capacity and storage technology. To show heterogeneity in provisioning can be extended to other resource types, we run two experiments on Amazon AWS, a public cloud provider. We run the YCSB-C [10] workload on a read-only MySQL database application deployed on iso-cost clusters of VM instances provisioned on AWS for different target arrival rates. ProxySQL [69] serves as the load balancer for the cluster running MySQL-5.7. The load generator and the load balancer are provisioned on separate m5.xlarge instances. The dataset of 2.5M records is pre-loaded on all the instances to serve read (SELECT) queries.

7.1 Memory capacity

To demonstrate the effects of different heterogeneity compositions when memory capacity is a key resource, we select the t3.large (2vCPU, 8GB memory) and t3.medium (2vCPU, 4GB memory) instances as the fast and frugal server types. While both the instances have similar compute capabilities, the fast server is 2× the cost and has twice the memory capacity over the frugal server. Thus, the frugal server has a higher service time than the fast server as requests now access storage more often. Figure 14a shows that a heterogeneous mix is effective in reducing the tail latency by about 11%-15% (across various load levels) as compared to a FrugalCluster while offering 50% more throughput than a FastCluster.



(a) Memory Capacity



(b) Storage systems

Figure 14: Heterogeneity arising out of other resource types running on AWS.

7.2 Storage system

To demonstrate the effects of different heterogeneity compositions when storage is a key resource, we consider the c5d.xlarge and a1.xlarge instances types. The c5d.xlarge is a Xeon processor server backed by a fast local NVMe storage device, and the a1.xlarge consists of the newly launched ARM processor servers backed by the comparatively slower Elastic Block Store (EBS). The fast c5d.xlarge instance is 1.8× more expensive than a frugal a1.xlarge instance.

As seen in Figure 14b, the heterogeneous mix of servers gives a better performance at the tail by about 15%-20% as compared to a FrugalCluster while offering significantly more throughput than a FastCluster.

These two experiments demonstrate the effectiveness of using heterogeneity as a design knob not just for compute but other resource types as well.

8 Related work

The related work can be classified into three categories: (i) solutions which design with heterogeneity, (ii) techniques which profit under incidental heterogeneity, (iii) systems which do not explicitly consider heterogeneity.

Design with heterogeneity: Heterogeneity is a powerful tool that has been previously applied in cloud for web applications and other data analytic applications with unique challenges and constraints.

In the cloud, researchers have studied heterogeneous designs using combinations of different VM instance types such as reserved instances (long term), on-demand instances (priced hourly), and spot instances [20, 34, 62, 68, 78, 81, 88, 91, 94]. Spot instances do not guarantee availability making them unsuitable for tail latency sensitive applications as unpredictable resource availability can be catastrophic on the performance. Extending our provisioning approaches to be aware of availability for user facing applications is a direction of future work. Reserved instances as advertised are best suited for steady-state usage [46]. They are cheaper when the cost is amortized over a long-term commitment. Typically [15], reserved instances are provisioned for the long-term steady state load and on-demand instances offering the same performance (homogeneous) are provisioned for the short-term time-varying load. Our solution complements these approaches especially when provisioning the on-demand instances. The equal cost allocation policy can be used to provision heterogeneous reserved instances and the load proportional cost allocation policy can be used for provisioning on-demand instances. Most of these works explore heterogeneity with respect to availability and cost rather than performance. By contrast, our work studies heterogeneity in the context of performance, where request scheduling and arriving at the correct heterogeneity composition become challenging problems.

Recently, [16] explores the scaling of a heterogeneous mix of big and small cores on a single node while considering the tail-latency performance under iso-cost configurations. These solutions improve the performance and/or energy efficiency of a single node, but do not address how to manage a cluster of nodes in the cloud. In these contexts, the cost for the node is already paid for upfront, and it is not possible to dynamically change the heterogeneity mix on the fly.

Incidental heterogeneity: Incidental heterogeneity is natural in a datacenter because of hardware upgrades occurring over time. [1, 26] deal with managing this on clusters for the map-reduce application by proposing efficient scheduling mechanisms for energy and performance. [35, 61] exploit the incidental heterogeneity to match varying resource requirements (in terms of CPU/memory) of different applications. Prior works [13–15, 56, 83] have also explored cluster scale job placement and scheduling to match jobs to (incidentally) heterogeneous hardware in order to maximize application performance or minimize performance interference between the applications. While these works focus on fixing the problems introduced by preexisting heterogeneity, our work shows a way to modulate heterogeneity to improve performance (e.g., minimize tail latency).

While heterogeneity aware load balancing like HALO [25] is generally a good idea, there are many scenarios which we specifically consider and address in our work. Furthermore, in contrast to our work, HALO does not consider the possibility of using heterogeneity as a tunable knob to maximize application performance.

Heterogeneity oblivious resource provisioning: Capacity provisioning of datacenters is a well studied area [2, 9, 28, 31, 32, 38, 64, 65, 86, 92, 93]. Contrasting to those approaches, which involve right-sizing capacities by forecasting demand over the time-scale of years, our work targets the cloud setting, which operates on the order of minutes. This leads to a different set of resource management problems. In the cloud, our work is more closely related to dynamic capacity provisioning (i.e., autoscaling) techniques. This is also a well studied area consisting of proactive and reactive approaches [22–24, 29, 33, 66, 80, 84, 87, 89] with sophisticated workload predictions [8, 29, 74], and only a few consider budget constraints (e.g., [59, 60]). All these autoscaling works do not consider heterogeneity and more importantly the associated scheduling challenges. Thus, our work improves upon this body of literature by showing how heterogeneity can be flexibly used as a knob for handling load variations and load mispredictions. To our knowledge this is the first paper to design and build clusters with heterogeneous mixes of servers to address tail latency under a fixed budget constraint.

9 Concluding remarks and future work

This paper proposes heterogeneity as an effective knob for meeting the time-varying scaling needs of web applications that have to operate within a specified budget over a given time horizon. While there are some load and/or budget extremes where homogeneous options turn out to be optimal, there is a fairly large middle ground where heterogeneous clusters can leverage the high service rates of fast servers and the scale-out benefits of frugal servers simultaneously to improve performance and make it more predictable.

Using queuing theory, we have studied a range of load aware and request size aware scheduling policies to gain insights on good policies for reducing tail latency. Based on these insights, we have built an actual heterogeneous cluster using real hardware both in our laboratory and on the public cloud, running various applications. Our implementation dynamically provisions heterogeneous resources over time and also includes a request scheduler that implements the policies we developed. We have evaluated this system to show that (i) heterogeneity is a much more nimble knob to adapt to a wide range of fluctuating loads than homogeneous options; (ii) heterogeneity provides a low and consistent tail latency despite zero knowledge of future load; (iii) with some load prediction, heterogeneity can help smooth out the effects of any mispredictions; and (iv) our central queue + work stealing policy works well to reduce tail latency even without request size knowledge. Thus, our proposed system provides consistently low tail latencies across a wide spectrum of load and budget availability regions.

In this work, we have considered a restricted class of applications – namely those that are easy to scale in a distributed fashion, incur low overheads for starting/stopping instances [15], and have insignificant costs associated with replication/restart. Although our applications do not share state, our solution can be extended to such applications by using prior works like [37]. We intend to extend our work to a wider spectrum of multi-tier stateful applications.

Acknowledgments

We thank anonymous reviewers for the constructive critique and useful feedback. This research was supported by National Science Foundation grants NSF-1909004, 1714389, 1629915, 1629129, 1526750, 1763681, 1912495 and a DARPA/SRC JUMP award.

References

[1] Faraz Ahmad, Srimat T Chakradhar, Anand Raghunathan, and TN Vijaykumar. 2012. Tarazu: optimizing mapreduce on heterogeneous clusters. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Vol. 40. ACM, 61–74.

[2] Douglas Alger. 2005. Choosing an optimal location for your data center. *Build the Best Data Center Facility for Your Business* (2005).

[3] AWS. 2018. AWS EC2 Pricing. https://aws.amazon.com/ec2/pricing/. [Online; accessed 20-Jan-2020].

[4] AWS. 2018. How Elastic Loadbalancing works. https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/how-elastic-load-balancing-works.html. [Online; accessed 20-Jan-2020].

[5] AWS. 2019. Amazon EC2 A1 Instances. https://aws.amazon.com/ec2/instance-types/a1/. [Online; accessed 20-Jan-2020].

[6] Mythic Beasts. 2019. Raspberry Pi. https://www.mythic-beasts.com/order/rpi. [Online; accessed 20-Jan-2020].

[7] Bespecular. 2018. Bespecular. https://www.bespecular.com/. [Online; accessed 20-Jan-2020].

[8] Rodrigo N Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. 2015. Workload prediction using ARIMA model and its impact on cloud applicationsâĂŹ QoS. *IEEE Transactions on Cloud Computing* 3, 4 (2015), 449–458.

[9] Byung-Gon Chun, Gianluca Iannaccone, Giuseppe Iannaccone, Randy Katz, Gunho Lee, and Luca Niccolini. 2010. An energy case for hybrid datacenters. *ACM SIGOPS Operating Systems Review* 44, 1 (2010), 76–80.

[10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 143–154.

[11] Jeffrey Dean and Luiz André Barroso. 2013. The tail at scale. *Commun. ACM* 56, 2 (2013), 74–80.

[12] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS operating systems review*, Vol. 41. ACM, 205–220.

[13] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, Vol. 48. ACM, 77–88.

[14] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 127–144.

[15] Christina Delimitrou and Christos Kozyrakis. 2016. Hcloud: Resource-efficient provisioning in shared cloud systems. *ACM SIGOPS Operating Systems Review* 50, 2 (2016), 473–488.

[16] Christina Delimitrou and Christos Kozyrakis. 2018. Amdahl's law for tail latency. *Commun. ACM* 61, 8 (2018), 65–72.

[17] Sheng Di, Derrick Kondo, and Walfredo Cirne. 2012. Host load prediction in a Google compute cloud with a Bayesian model. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 21.

[18] Diego Didona and Willy Zwaenepoel. 2019. Size-aware Sharding For Improving Tail Latencies in In-memory Key-value Stores.. In *NSDI*. 79–94.

[19] Digicube. 2019. Digicube. http://digicube.fr/rapidserveurs. [Online; accessed 20-Jan-2020].

[20] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D Bowers, and Michael M Swift. 2012. More for your money: exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 20.

[21] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. 2009. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28, 13 (2009), 2009.

[22] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. 2014. Adaptive, Model-driven Autoscaling for Cloud Applications.. In *ICAC*, Vol. 14. 57–64.

[23] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. 2012. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)* 30, 4 (2012), 14.

[24] Anshul Gandhi, Sidhartha Thota, Parijat Dube, Andrzej Kochut, and Li Zhang. 2016. Autoscaling for hadoop clusters. In *Cloud Engineering (IC2E), 2016 IEEE International Conference on*. IEEE, 109–118.

[25] Anshul Gandhi, Xi Zhang, and Naman Mittal. 2015. HALO: Heterogeneity-Aware Load Balancing. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*. IEEE, 242–251.

[26] Rohan Gandhi, Di Xie, and Y Charlie Hu. 2013. PIKACHU: How to Rebalance Load in Optimizing MapReduce On Heterogeneous Clusters.. In *USENIX Annual Technical Conference*. 61–66.

[27] Garth A Gibson and Rodney Van Meter. 2000. Network attached storage architecture. *Commun. ACM* 43, 11 (2000), 37–45.

[28] Inigo Goiri, Kien Le, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. 2011. Intelligent placement of datacenters for internet services. In *International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 131–142.

[29] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*. Ieee, 9–16.

[30] Google. 2018. GRPC Framework. https://grpc.io/. [Online; accessed 20-Jan-2020].

[31] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. 2008. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review* 39, 1 (2008), 68–73.

[32] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. 2009. VL2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, Vol. 39. ACM, 51–62.

[33] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S McKinley, and Björn B Brandenburg. 2017. Swayam: distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM, 109–120.

[34] Tian Guo, Upendra Sharma, Timothy Wood, Sambit Sahu, and Prashant Shenoy. 2012. Seagull: intelligent cloud bursting for enterprise applications. In *Proceedings of the 2012 USENIX conference on Annual Technical Conference*. USENIX Association, 33–33.

[35] Vishal Gupta, Min Lee, and Karsten Schwan. 2015. Heterovisor: Exploiting resource heterogeneity to enhance the elasticity of cloud platforms. *ACM SIGPLAN Notices* 50, 7 (2015), 79–92.

[36] Vishal Gupta and Karsten Schwan. 2013. Brawny vs. wimpy: Evaluation and analysis of modern workloads on heterogeneous processors. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*. IEEE, 74–83.

[37] Ubaid Ullah Hafeez, Muhammad Wajahat, and Anshul Gandhi. 2018. ElMem: Towards an Elastic Memcached System. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 278–289.

[38] Ronny Hans, Ulrich Lampe, and Ralf Steinmetz. 2013. QoS-aware, cost-efficient selection of cloud data centers. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 946–947.

[39] Mor Harchol-Balter. 2013. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press. 274–275 pages.

[40] Mor Harchol-Balter, Mark E Crovella, and Cristina Murta. 1998. On choosing a task assignment policy for a distributed server system. (1998).

[41] Mor Harchol-Balter, Alan Scheller-Wolf, and Andrew R Young. 2009. Surprising results on task assignment in server farms with high-variability workloads. *ACM SIGMETRICS Performance Evaluation Review* 37, 1 (2009), 287–298.

[42] Mor Harchol-Balter, Bianca Schroeder, Nikhil Bansal, and Mukesh Agrawal. 2003. Size-based scheduling to improve web performance. *ACM Transactions on Computer Systems (TOCS)* 21, 2 (2003), 207–233.

[43] Herodotos Herodotou, Fei Dong, and Shivnath Babu. 2011. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 18.

[44] Yu-Ju Hong, Jiachen Xue, and Mithuna Thottethodi. 2011. Dynamic server provisioning to minimize cost in an IaaS cloud. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. ACM, 147–148.

[45] Hound. 2008. Hound App by SoundHound Inc. https://soundhound.com/hound. [Online; accessed 20-Jan-2020].

[46] Amazon Web Services Inc. 2019. Amazon EC2 Reserved Instances. https://aws.amazon.com/ec2/pricing/reserved-instances/. [Online; accessed 20-Jan-2020].

[47] Amazon Web Services Inc. 2019. New âĂŞ Savings Plans for AWS Compute Services. https://aws.amazon.com/blogs/aws/new-savings-plans-for-aws-compute-services/. [Online; accessed 20-Jan-2020].

[48] NVM Express Inc. 2018. NVM Express. https://nvmexpress.org/. [Online; accessed 20-Jan-2020].

[49] Uptime Institute. 2018. Data center industry survey results. https://uptimeinstitute.com/2018-data-center-industry-survey-results. [Online; accessed 20-Jan-2020].

[50] Intel. 2018. Intel Atom Processors. https://www.intel.com/content/www/us/en/products/processors/atom.html. [Online; accessed 20-Jan-2020].

[51] Intel. 2018. Intel Xeon Processors. https://www.intel.com/content/www/us/en/products/processors/xeon.html. [Online; accessed 20-Jan-2020].

[52] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters.. In *OSDI*. 117–134.

[53] Harshad Kasture and Daniel Sanchez. 2016. TailBench: A benchmark suite and evaluation methodology for latency-critical applications. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*. IEEE, 1–10.

[54] Andrew Krioukov, Prashanth Mohan, Sara Alspaugh, Laura Keys, David Culler, and Randy H Katz. 2010. Napsac: Design and implementation of a power-proportional web cluster. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*. ACM, 15–22.

[55] Paul Lamere, Philip Kwok, Evandro Gouvea, Bhiksha Raj, Rita Singh, William Walker, Manfred Warmuth, and Peter Wolf. 2003. The CMU SPHINX-4 speech recognition system. In *IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2003), Hong Kong*, Vol. 1. 2–5.

[56] Gunho Lee and Randy H Katz. 2011. Heterogeneity-Aware Resource Allocation and Scheduling in the Cloud.. In *HotCloud*.

[57] Pejman Lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer, et al. 2012. Scale-out processors. In *ACM SIGARCH Computer Architecture News*, Vol. 40. IEEE Computer Society, 500–511.

[58] Myron H MacDougall. 1987. *Simulating computer systems: techniques and tools*. MIT press.

[59] A Hasan Mahmud, Yuxiong He, and Shaolei Ren. 2015. BATS: budget-constrained autoscaling for cloud performance optimization. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, 232–241.

[60] Ming Mao, Jie Li, and Marty Humphrey. 2010. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*. IEEE, 41–48.

[61] Jason Mars and Lingjia Tang. 2013. Whare-map: heterogeneity in homogeneous warehouse-scale computers. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Vol. 41. ACM, 619–630.

[62] Ishai Menache, Ohad Shamir, and Navendu Jain. 2014. On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud.. In *ICAC*. 177–187.

[63] Mininodes. 2019. Raspberry Pi 3 Server. https://www.mininodes.com/product/raspberry-pi-3-server/. [Online; accessed 20-Jan-2020].

[64] Iyswarya Narayanan, Aman Kansal, and Anand Sivasubramaniam. 2017. Right-Sizing Geo-distributed Data Centers for Availability and Latency. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 230–240.

[65] Iyswarya Narayanan, Aman Kansal, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Sriram Govindan. 2014. Towards a Leaner Geo-distributed Cloud Infrastructure.. In *HotCloud*.

[66] Pradeep Padala, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. 2007. Adaptive control of virtualized resources in utility computing environments. In *ACM SIGOPS Operating Systems Review*, Vol. 41. ACM, 289–302.

[67] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A Kozuch, and Gregory R Ganger. 2018. 3sigma: distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2.

[68] Eric Pettijohn, Yanfei Guo, Palden Lama, and Xiaobo Zhou. 2014. User-Centric Heterogeneity-Aware MapReduce Job Provisioning in the Public Cloud.. In *ICAC*. 137–143.

[69] ProxySQL. 2018. ProxySQL. https://proxysql.com. [Online; accessed 20-Jan-2020].

[70] Will Reese. 2008. Nginx: the high-performance web server and reverse proxy. *Linux Journal* 2008, 173 (2008), 2.

[71] Harvard Business Review. 2011. IT Project. https://hbr.org/2011/09/why-your-it-project-may-be-riskier-than-you-think. [Online; accessed 20-Jan-2020].

[72] Eduardo Roloff, Matthias Diener, Emmanuell Diaz Carreño, Luciano Paschoal Gaspary, and Philippe OA Navaux. 2017. Leveraging cloud heterogeneity for cost-efficient execution of parallel applications. In *European Conference on Parallel Processing*. Springer, 399–411.

[73] Eduardo Roloff, Matthias Diener, Emmanuell D Carreño, Francis B Moreira, Luciano P Gaspary, and Philippe OA Navaux. 2017. Exploiting Price and Performance Tradeoffs in Heterogeneous Clouds. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. ACM, 71–76.

[74] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 500–507.

[75] Daniel Schall and Volker Hudlet. 2011. Wattdb: an energy-proportional cluster of wimpy nodes. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 1229–1232.

[76] SenseTime. 2018. SenseTime. https://www.sensetime.com/. [Online; accessed 20-Jan-2020].

[77] Amazon Web Services. 2018. AWS Elastic Block Store. https://aws.amazon.com/ebs/. [Online; accessed 20-Jan-2020].

[78] Prateek Sharma, Stephen Lee, Tian Guo, David Irwin, and Prashant Shenoy. 2015. Spotcheck: Designing a derivative iaas cloud on the spot market. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 16.

[79] Shazam. 2009. Shazam App. https://www.shazam.com/. [Online; accessed 20-Jan-2020].

[80] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 5.

[81] Supreeth Subramanya, Tian Guo, Prateek Sharma, David Irwin, and Prashant Shenoy. 2015. Spoton: a batch computing service for the spot market. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 329–341.

[82] Byung-Chul Tak, Bhuvan Urgaonkar, and Anand Sivasubramaniam. 2011. To Move or Not to Move: The Economics of Cloud Computing.. In *HotCloud*.

[83] Alexey Tumanov, Timothy Zhu, Jun Woo Park, Michael A Kozuch, Mor Harchol-Balter, and Gregory R Ganger. 2016. TetriSched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 35.

[84] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, and Pawan Goyal. 2005. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*. IEEE, 217–228.

[85] Erik-Jan van Baaren. 2009. Wikibench: A distributed, wikipedia based web application benchmark. *Master's thesis, VU University Amsterdam* (2009).

[86] Kashi Venkatesh Vishwanath, Albert Greenberg, and Daniel A Reed. 2009. Modular data centers: how to design them?. In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*. ACM, 3–10.

[87] Muhammad Wajahat, Alexei Karve, Andrzej Kochut, and Anshul Gandhi. 2017. MLscale: A Machine Learning based Application-Agnostic Autoscaler. *Sustainable Computing: Informatics and Systems* (2017).

[88] Cheng Wang, Bhuvan Urgaonkar, Aayush Gupta, George Kesidis, and Qianlin Liang. 2017. Exploiting Spot and Burstable Instances for Improving the Cost-efficacy of In-Memory Caches on the Public Cloud. In *Proceedings of the Twelfth European Conference on Computer Systems*. ACM, 620–634.

[89] Ji Wang, Weidong Bao, Xiaomin Zhu, Laurence T Yang, and Yang Xiang. 2015. FESTAL: fault-tolerant elastic scheduling algorithm for real-time tasks in virtualized clouds. *IEEE Transactions On Computers* 64, 9 (2015), 2545–2558.

[90] Qiumin Xu, Huzefa Siyamwala, Mrinmoy Ghosh, Tameesh Suri, Manu Awasthi, Zvika Guz, Anahita Shayesteh, and Vijay Balakrishnan. 2015. Performance analysis of NVMe SSDs and their implication on real world databases. In *Proceedings of the 8th ACM International Systems and Storage Conference*. ACM, 6.

[91] Zichen Xu, Christopher Stewart, Nan Deng, and Xiaorui Wang. 2016. Blending on-demand and spot instances to lower costs for in-memory storage. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 1–9.

[92] Guihai Yan, Jun Ma, Yinhe Han, and Xiaowei Li. 2016. EcoUp: Towards Economical Datacenter Upgrading. *IEEE Transactions on Parallel and Distributed Systems* 27, 7 (2016), 1968–1981.

[93] Sungkap Yeo and Hsien-Hsin Lee. 2011. Using mathematical modeling in provisioning a heterogeneous cloud computing environment. *Computer* 44, 8 (2011), 55–62.

[94] Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo. 2015. Exploiting cloud heterogeneity to optimize performance and cost of mapreduce processing. *ACM SIGMETRICS Performance Evaluation Review* 42, 4 (2015), 38–50.