Join the Best Queue: Reducing Performance Variability in Heterogeneous Systems

***Abstract*—Cloud computing is an emerging paradigm, featuring on-demand provisioning of virtual machines (VMs) with different sizes and images. Systems deployed in the cloud can thus be heterogeneous, i.e., using multiple different types of VM instances. The immediate performance challenge arises: how to best use heterogeneous VM instances such that the experienced performance, i.e., response times, on different VMs are similar. In this paper, we first show to what extent deployed systems are heterogeneous in clouds by collecting data from operational data centers. We also show that response times suffer from high variance across replicas hosted on different VMs. We develop a novel plug&play workload controller, Join-the-Best-Queue (JBQ), which aims to reduce the variance and higher percentile of response-times in heterogeneous environments. With the aid of a testbed hosting part of wikipedia in an operational cloud, we show that JBQ can significantly reduce performance variability across different VMs, compared to prevailing load distributing policies in the Apache web server.**

***Keywords*-Load control algorithms; heterogeneous systems; cloud computing**

I. Introduction

Hosting services and applications in cloud environments offers many advantages, such as on-demand provisioning of resources and ease of management costs. In an Infrastructure-as-a-Service cloud, users can lease the necessary computing resources in terms of Virtual Machines (VMs), which are dimensioned in multiple specifications of virtual CPUs and memory, bundled with a selection of different pre-configured and user-defined virtual images. The underlying hardware of clouds exhibits a high degree of diversity, e.g., CPU cores and RAM per machine [20], and workloads executed there are very dynamic and time-varying, featuring many short jobs. Consequently, resource management is by no means easy in the cloud, which is populated with heterogeneous physical and virtual resources.

The immediate performance challenges of hosting applications in clouds are how to select different VM configurations, e.g., small, medium, and large instances in Amazon EC2 [6], and how to efficiently assign jobs to heterogeneous VM instances. In both cases, one relies on the knowledge of computational capacities of VMs executing one's applications. Unfortunately, it is very difficult to estimate the "power" of VMs in an unknown heterogeneous environment hosting dynamic workloads. Consequently, a common performance pitfall of cloud computing, highlighted by several studies [5, 9, 12] working at the execution of different applications, is high performance variability due to the heterogeneity in the cloud. For example, VMs with the same configuration may yield very different response times for the same workload. Highly varying response times can lead to a serious worst performance scenario, such as a high value of the 95^{th} percentile of response time, and lead to a fairness problem among the requests being served.

There are a large number of load control algorithms aimed at reducing response times, developed for homogeneous systems, such as the byrequest and bybusyness policies in Apache web server [3]. In a cluster of multiple VMs hosting replicated services, they tend to assume that the computational capacities of all VMs are the same and dispatch similar amounts of requests to them. Such policies can result in a high variance of response times and potentially worse higher percentile of response time. For heterogeneous clusters, several algorithms [13, 23] aim at minimizing the expected response time, assuming that computational capacities of all servers are known and constant. To the best of our knowledge, there is no clear solution on how to estimate the computational capacity in a complex environment like the cloud, where intrusive profiling on VMs comes at a high cost.

In this paper, we aim to answer two research questions: (1) how heterogenous are typical systems deployed in a cloud, given a number of different VM configurations? (2) how can cloud users reduce performance variability, i.e., the variance and high percentile of response times when hosting applications on a cluster of heterogenous VMs? To this end, we collect user statistics from operational clouds and characterize the heterogeneity of users' systems by the number and types of VMs. To ensure consistent response times with low variance in a heterogenous VM cluster, we develop the plug&play load-controlling algorithm Join-the-Best-Queue (JBQ), which dispatches requests to the VM with the shortest expected response time using the estimated computational capacities of different VMs in an on-line and non-intrusive fashion. The required inputs for JBQ are simply the number of outstanding requests and the inter-departure time of requests. We implement and evaluate JBQ on several clusters of wikepedia system in a production cloud and show that JBQ can indeed detect the computational capacities of VMs and reduce the variance of response times across VMs, with only a slight increase in average response time compared to prevailing load control algorithms in Apache web server.

The scientific contributions of this paper are three-fold:

1) We present a small-scale characterization study on the heterogeneity of users' deployed systems in clouds.

2) We develop the novel load-controlling algorithm JBQ that is particularly well suited to ensure consistent response times with low variance when hosting applications on a heterogeneous cluster.
3) We perform a thorough evaluation of the JBQ algorithm in an operational cloud, comparing it with the state-of-the-art load control algorithm bybusyness in Apache web server. In particular, we show that JBQ significantly outperforms bybusyness with respect to variance and the values of the higher percentiles of response times.

This paper is organized as follows: The heterogeneity of users' systems and resulting performance issues are explained in Section II. The proposed JBQ algorithm is described in Section III. Section IV contains the experimental results. Related studies are summarized in Section V. Section VI concludes this paper.

II. Heterogeneity and Performance Issues

In this section, we try to understand what kind of system is typically deployed by users in clouds, pertaining to the heterogeneity of VM instances, using the data collected from two operational data centers. We then highlight the performance challenges arising in such a heterogeneous environment, i.e., response times vary drastically from one VM instance to another, for a cluster of different types of VM instances and even for the same type of instances.

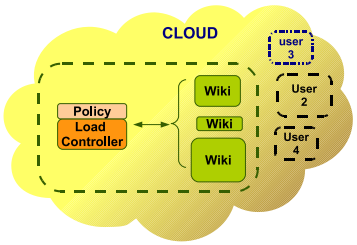


Figure 1: System schematics of a small cluster of VMs with heterogeneous sizes.

A. Heterogeneous Systems

We collect statistics about users' systems deployed in clouds, i.e., the size of such systems and their composition of VM types. The observed cloud systems are fully operational with a total of 142 users and offer three types of VM instances: gold, silver, and bronze. The specifications of those instances are listed in Table I. There are four standard images available, namely Fedora, Red Hat, OpenSuse, and CentOS. In summary, there is a diverse set of systems in the observed clouds.

Fig. 2 summarizes the characteristics of systems deployed in the cloud. Fig. 2 (a) shows the distribution of the number of VM instances per system. One can see that more than 75% of the users deploy more than one VM. And, the majority of the users' systems comprise two to five VMs. Moreover, Fig. 2 (b) shows the composition of different types of instances per system. Bronze, silver, and gold represent the systems consisting of a single type of VM instances only, whereas mixed denotes the systems with mixed types of VM instances. At first glance, one can see that 35% of the users' systems use heterogeneous VM types, while the rest of the systems are composed of identical VMs.

Table I: Specifications of VM instances in the cloud.

	RAM [GB]	Processing Units	Price[cent/hour]
Bronze	2	1	3
Silver	2	2	6
Gold	4	4	12

However, as pointed out by several empirical studies [10, 12, 22], VM instances of the same type may show very different performance under the same load condition, due to unknown neighboring VMs hosted on the same physical machine. To further confirm the performance heterogeneity of the same VM type, we execute a wikipedia [24] workload on multiple instances of silver VMs, each of which receives an average number of 3.69 requests per second and has a CPU utilization of around 85%. The detailed setup is explained in Section IV. Fig. 3 (a) and (b) depict the average response times and the throughput, respectively. One can observe that although each VM has the same specification and load condition, the response times and throughput are very different from one instance to another.

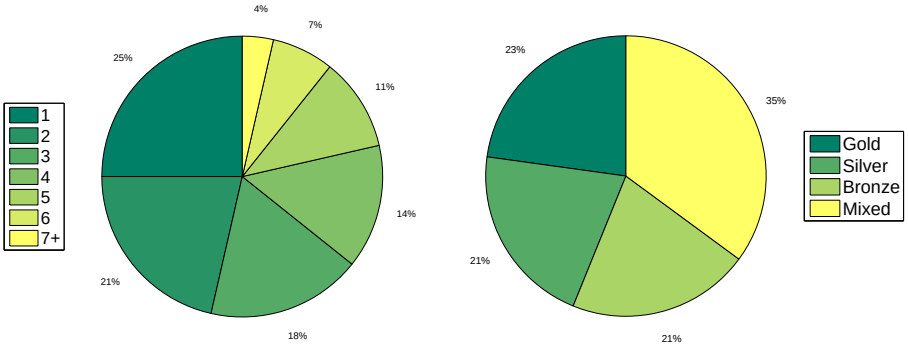
In summary, we conclude that systems deployed in clouds are composed of VM instances with heterogeneous computational capacities, which results not only from different types of VMs but also from the same type of VMs, a hidden heterogeneity in the cloud.

B. Performance Challenges in Heterogeneous Environments

We highlight the performance challenges when deploying replicated web services in a cloud environment consisting of instances of heterogeneous VMs. We deploy wikipedia on a cluster consisting of three VMs, one gold, one silver, and one bronze in an operational cloud. In addition, our deployment scenario includes a fourth VM of type bronze, which hosts a load generator producing client requests as well as a load controller.

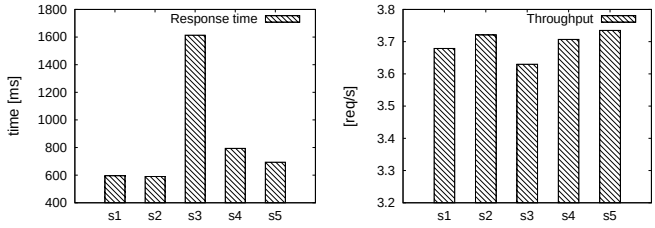
Figure 1 depicts the deployment setting in our experiment. The load controller uses the bybusyness policy, one of the default load control algorithms available on Apache web server [3]. This scheme tries to dispatch client requests to the VM with the lowest number of outstanding requests. The bybusyness policy is almost identical to JSQ, except when it has to handle the situation of multiple queues with the same number of outstanding requests.

We summarize the response times of each VM over a



(a)Number of instances per system (b) Types of VM instances per system

Figure 2: The characteristics of systems deployed in the cloud: heterogeneity.



(a) Response time (b) Throughput

Figure 3: Hidden heterogeneity of silver VMs in the cloud.

45 minute interval in Fig. 4. One can see that the variance of response times across VMs is very high; the average response times for gold, silver, and bronze VMs are 400 ms, 600 ms, and 1400 ms, respectively. The bronze VM serves 6663 requests, that is 16.65% of the total number of requests. Hence, a non-negligible percentage of client requests are processed with high response times, possibly violating service-level agreements that are often defined by the worst performance, i.e., 95^{th} percentile of response time should be less than a certain target value.

To understand the cause of such a phenomenon, we collect CPU utilization on the three VMs, which shows very different values. This observation implies that the bronze VM receives more client requests than its capacity, whereas the gold VM could have processed more requests with only a negligible increase in response times. The bybusyness policy seems unable to dispatch requests according to the VMs' computational capacities. Therefore, we conjecture that the number of outstanding requests is not a good indicator of the VMs' capacities in a heterogeneous environment. Moreover, the fact that even VMs of the same type may have significantly different computational capacities further exacerbates the difficulty of finding a VM that can achieve the lowest response time upon the arrival of a request.

The aforementioned observations lead us to ask the question how one should dispatch requests in a heterogeneous system hosting replicated services, such that performance variability across VMs as well as the the worst performance are minimized. The answer hinges on a challenging task: finding, preferably in a non-intrusive way, a metric indicating the computational capacity of VMs. In the following section, we propose a simple plug&play load-control algorithm, namely Join-the-Best-Queue (JBQ), which tries to find the best VM that can minimize the variability and worst case of response times across VMs.

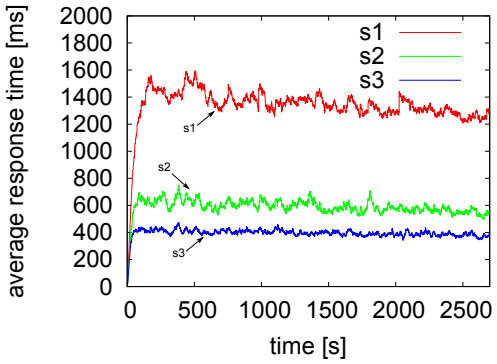


Figure 4: Response times of VMs under the bybusyness policy.

III. JBQ: A Workload Controller for Heterogeneous Systems

In this section, we develop a plug&play load controller, using the proposed policy. JBQ aims to reduce the variance of response times across VMs and the worst-case scenarios, i.e., higher percentile of response times. To capture the computational capacity of each VM, JBQ relies only on the information about the number of outstanding requests and inter-departure times between requests, without any intrusive profiling on VM instances. In the following text, we first introduce a methodology to estimate computational capacities of VMs and then detail two main steps of JBQ: pick up the best queue and handle the request completion.

A. Estimation of Computational Capacity

Join the queue with the shortest expected delay (SED) [13, 23] is a well-known greedy policy to reduce users' response times in systems with multiple homogeneous and heterogeneous servers. In the former case, SED policy is essentially equivalent to the Join-the-Shortest-Queue (JSQ) policy, also referred to as bybusyness in the Apache web server. In a heterogeneous system consisting of multiple servers with a single processing unit, it dispatches requests to the server which has the shortest expected delay, defined as the current queue length plus the arriving request, denoted by $Q + 1$, divided by the service rate of each server, denoted by μ

$$\frac{Q+1}{\mu}, \tag{1}$$

where the service rate is defined by the inverse of processing time. However, in a cloud environment, where the computational capacity of each VM is not known and even possibly changes over time, it is very challenging to estimate the processing time of requests because an intrusive profiling on VMs is often required [5]. Moreover, as the processing time of a single request depends on the level of parallelism of the application [1], the service rate, the inverse of processing time, used in Eq. 1 does not truly reflect the computational capacities in VMs with multiple processing units.

A naive solution to estimate the computational capacity of modern VMs with multiple processing units would be using the throughput of the VMs. To obtain the throughput, one can simply compute the average of the inter-departure times of the requests, denoted by T, and then compute the throughput as the inverse of the average inter-departure time, $X = \frac{1}{T}$. The drawback of such an approach is that the throughput even includes the fraction of time when the VM is in an idle state. Consequently, the throughput may only indicate the deflated computational capacity of a VM, depending on the existence of idle states.

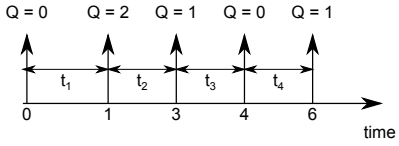


Figure 5: Illustration of the departure of five requests. Q is the number of outstanding requests of the VM at the departure time and t is the inter-departure time.

Therefore, to accurately estimate the computational capacities of VMs via throughput, we propose collecting only the inter-departure times, which do not include the idle times, namely *valid inter-departure times*, when computing the inverse of average inter-departure times. We illustrate this idea with an example in Fig. 5. There are 5 departures and consequently 4 inter-departure times. If at the beginning of an inter-departure time the queue length is at at zero, $Q = 0$, this indicates that the VM turns idle as there are no more requests in the system. Therefore, $t1$, $t4$ are inter-departure times that include idle times, whereas $t2$ and $t3$ are so-called valid inter-departure times. Therefore, we define the valid inter-departure times as the ones with non-zero queue length at the beginning.

Finally, we propose estimating the computational capacity of a VM, denoted as Φ, as the inverse of the average valid inter-departure times, T_{valid}. Thus, we write a computational capacity of VM i, $i \in \{1 \dots N\}$, as

$$\Phi_i = \frac{1}{T_i^{valid}}. \tag{2}$$

Moreover, to accommodate fluctuation of computational capacities of VMs in the cloud, we apply Exponentially Weighted Moving Average (EWMA) to update T_i^{valid} upon every valid inter-departure time. We let k_i denote the number of updates of VM i, and compute the EWMA estimate of $T_i^{valide}(k_i)$ as follows:

$$T_i^{valid}(k_i) = (1-\gamma)T_i^{valid}(k_i - 1) + \gamma t_i(k_i), \tag{3}$$

where γ is the forgetting factor, ranging between $[0, 1]$ and $t_i(k_i)$ is the time elapsed between the departure that triggers the update and the previous departure, e.g., $t2$ and t_3 in Fig. 5. A high value of γ is suitable for fast-changing workloads, whereas a low value of γ is appropriate for low-changing workloads. In our implementation we chose $\gamma = 0.1$.

B. Join the Best Queue (JBQ)

There are two main steps in the proposed JBQ algorithm: (1) upon a request arrival, computing the expected response time using computational capacities Φ_i of all available VMs and (2) upon a request departure, updating the computational capacity accordingly. In particular, JBQ applies the following formula to compute estimated response times for each VM i:

$$\hat{R}_i = \frac{1+Q_i}{\Phi_i}, \tag{4}$$

where Q_i is the observed number of outstanding requests of VM i when a request arrives and Φ_i is the latest value of estimated computational capacity.

The algorithm dispatches the request to VM i with the lowest estimated response time. To reduce the computational complexity of Eq. 4 we use $\frac{1}{T_i^{valid}}$ instead of Φ_i, and obtain the following formula to estimate the response time:

$$\hat{R}_i = (1+Q_i)T_i^{valid}, \tag{5}$$

whose computational complexity is lower than directly using the computational capacity in Eq. 4.

Algorithm 1 summarizes the implementation of the algorithm described above. The function $pickTheBestQueue$ is invoked for each request arrival, and the function

$handleRequestCompletion$ is invoked for each requestw departure.

Algorithm 1 The two steps of JBQ algorithm

```
function pickTheBestQueue()
    R ← {(1 + Q_i)T_i^valid, ..., (1 + Q_N)T_N^valid}
    res ← i s.t. R_i = min(R), i ∈ {1...N}
    Q_res ← Q_res + 1
    return res
end function

function handleRequestCompletion(i)
    if isNotIdle(i) then
        t_i ← current_time() − last_departure_i
        T_i^valid ← (1 − γ)T_i^valid + γt_i
    end if
    Q_i ← Q_i − 1
    if Q_i = 0 then
        setIdle(i, true)
    else
        setIdle(i, false)
        last_departure_i ← current_time()
    end if
end function
```

We highlight here that all the input parameters are collected at the load controller without any interaction with VMs, and without any profiling on the execution status of VMs. Therefore, JBQ is a light-weight and non-intrusive load-controlling policy.

IV. Evaluation

We implement the proposed JBQ policy on Apache web server 2.4.3 to reduce variance of response times for heterogeneous systems hosting wikipedia in the cloud. We benchmark JBQ against the bybusyness policy available in Apache web server, on three different types of clusters, i.e., three and eight VMs of the mixed type and ten instances of silver type. We refer to them as small, medium, and silver clusters. The specific performance metrics evaluated are the average, standard deviation, 95^{th}, 97^{th}, and 99^{th} percentile of response times.

A. Implementation

To build the wikipedia system, we downloaded the *pages-articles.xml* dump on October 12, 2012, and extracted a subset of 200000 entries. The system is composed of one load controller and N replicas of wikipedia, all hosted on VMs in an operational cloud, as shown in Fig. 1.

The VM of the load controller is equipped with the Fedora 14 operating system [7] and with the Apache web server 2.4.3 [3]. The other VMs are equipped with Fedora 14, Apache web server 2.4.2, the PHP 5.4.8 server-side script engine [19], MediaWiki 1.19.2 [16] as web application, and the MySQL 5.1.60 database [17]. To eliminate performance disturbance caused by the network between the clients and the load controller, both the workload generator and the load controller are deployed on the same VM.[1]

Fig. 6 illustrates the two-step communication between the client and the systems. Clients' requests are generated using JMeter 2.8 [2]. To stress the system, the workload generator uses the *Random article* service offered by the MediaWiki software. Upon receiving a random page request, the wiki system replies an HTTP redirect response to notify the client of the URL of the selected article. At the second step, the client requests the resource indicated by the URL and the server provides its contents. During this interaction, the workload generator communicates directly with the load controller. In general, the server generating an HTTP redirect reply may be different from the the server providing the article. Note that herein we focus on the interaction of the second step. As such, the response time presented in the following section is defined as the time spent in the second step, excluding the first step, which is almost negligible.

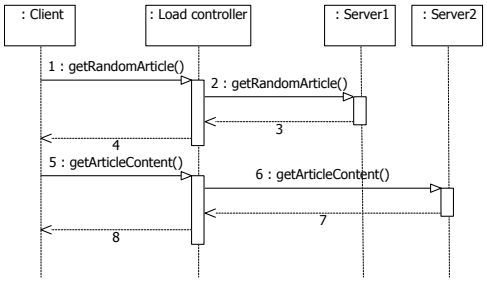


Figure 6: Communication between workload generator and system.

While we explore all the load control policies implemented in the Apache web server, in this paper we only present the comparison against Apache's bybusyness policy (similar to JSQ), because all other policies yielded significant inferior results. Therefore, our baseline for comparison is the policy of the Apache web server that gives the best results in our setting.

We chose Apache web server as implementation platform due to its prevalence in today's systems. Nevertheless, the proposed algorithm can easily be ported on other platforms since it just needs to run two simple functions (Algorithm 1) when a request arrives and leaves the cluster. Moreover, the algorithm has very low requirements in terms of memory.

B. A Small-Sized Cluster

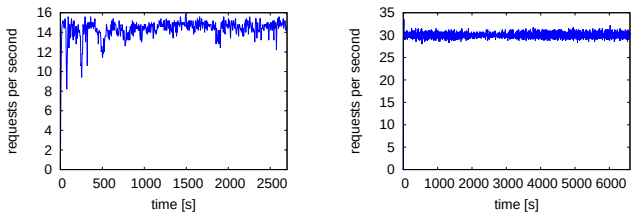
We deploy a small-size wikipedia cluster, consisting of three replicas on gold, silver, and bronze VM, respectively. Each experiment lasts for about 45 minutes and requires

[1]This VM does not become a bottleneck in any of our experiments, its CPU utilization remains below 10%.

Table II: Comparison between JBQ and bybusyness on different clusters: various statistics about response times.

	small cluster			medium cluster			silver cluster		
	byb [ms]	JBQ [ms]	$\frac{byb-JBQ}{byb}$ [%]	byb [ms]	JBQ [ms]	$\frac{byb-JBQ}{byb}$ [%]	byb [ms]	JBQ [ms]	$\frac{byb-JBQ}{byb}$ [%]
Mean	625.72	755.27	−20.7	439.91	449.90	−2.3	413.59	431.11	−4.23
St.dev	476.10	385.10	+19.1	249.93	202.84	+18.8	191.23	185.73	+2.87
99^{th}	2382.10	2018.02	+15.2	1383.97	1154.92	+16.6	1117.42	1131.94	−1.29
97^{th}	1706.59	1594.03	+6.6	1006.01	912.40	+9.30	856.08	867.09	−1.28
95^{th}	1467.11	1411.85	+3.7	872.66	812.56	+6.9	746.03	763.27	−2.31

the processing of 40000 client requests. The average size of the requests processed by the cluster is 43936 bytes. Fig. 7 (a) shows the requests per second sent by the workload generator. The arithmetic mean is around 14.3 requests per second. One can observe that the workload is very bursty.



(a) small cluster (b) medium-sized cluster

Figure 7: Workload in number of requests per second.

We summarize the response times under JBQ and bybusyness (abbreviated as byb) in the small cluster column of Table II. JBQ outperforms bybusyness in terms of standard deviation and higher percentile of response times, at the cost of a higher average response time. JBQ particularly achieves better results for the worst-case scenario, i.e., 99^{th} percentile response times. On the other hand, bybusyness results in highly varying response times. Such a phenomenon may not only lead to the violation of service-level agreements defined by the worst performance, but also introduces unfairness in request handling.

To better illustrate the difference between JBQ and bybusyness, in Fig. 8 we plot the average response times of each replica under JBQ. In order to enhance the readability of measurement results of response times, we apply an exponential weighted moving average with a weight of 0.01. Compared to bybusyness results summarized in Fig. 4, one can clearly see that JBQ sends a smaller number of requests to the bronze VM (5918) than to the silver (13072) and gold (21010) VMs. On the other hand, though bybusyness sends 6663 requests to the bronze, 14432 requests to the silver, and 18905 requests to the gold VM, it is not able to exploit the real computational capacity of the VMs, as illustrated in Fig. 4. As such, the requests processed by gold and silver VMs have significantly lower response times than the bronze VM, which in turn results in those higher percentile of response times. In Fig. 8 there is an interesting phenomenon around the 2500 second time in that the bronze VM instance exhibits an evident drop in the response time. We suppose that this is due to the fact the JBQ underestimated the computational capacity of the bronze instance. This situation does not cause any increase in the average response time in the silver and gold instances, unlike with bybusyness.

The aforementioned observations lead us to conclude that JBQ indeed distributes the requests to heterogeneous VMs according to their computational capacity and therefore the variance and higher percentile of response times can be mitigated. Nonetheless, a tradeoff of higher response times at more powerful VMs is unavoidable, compared to prevailing policies, e.g., bybusiness, which are oblivious to different computational capacities on heterogeneous VMs.

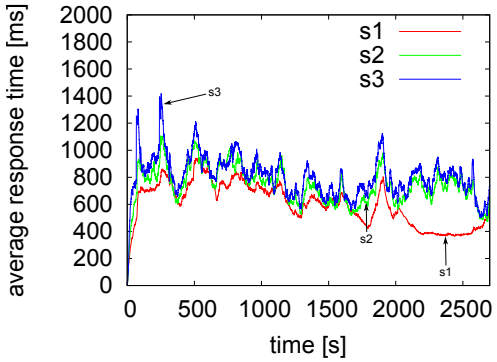


Figure 8: Response times of three VMs under JBQ policy: s1 (bronze), s2 (silver), s3 (gold).

C. Medium-sized Cluster

Herein, we evaluate JBQ on a medium-sized cluster, consisting of two bronze VMs, three gold VMs, and three silver VMs, denoted by s1, s2, s3, s4, s5, s6, s7, and s8, respectively. In addition to a larger cluster, we also extend the duration of the experiment to 2 hours with a total of 200000 requests. The request rate per second is plotted in Fig. 7 (b) and the average number of requests received by the wiki system is 29.96 per second.

We summarize the evaluation results against the bybusiness policy in the medium cluster column of Table II. Similar to the small cluster, JBQ can reduce the variance and higher percentile of response times, compared to the bybusiness. Another interesting observation is that the difference in average response times between JBQ and bybusyness is

decreased from -20% in the smaller cluster to -2.3% here. In a larger cluster with heterogeneous VMs, the diversity of VM computational capacities is higher, and thus bybusyness being oblivious to such a difference results in inferior performance, i.e., a higher value for average response time and the worst performance.

Moreover, we summarize the detailed statistics of each VM, i.e., the average response time (R) and received request rate (λ) in Table III. The request rates received at bronze VMs (S_1 and S_2) are much lower under JBQ, compared to bybusyness; therefore their response times twice as high as JBQ. Although bybusyness sends a lower number of request to bronze VMs than to silver and gold VMs, the resulting loads are still higher on other VMs and so are the response times. Moreover, bybusyness makes almost no differentiation between silver and gold VMs, in terms of request rates. As such, the response times on the gold VMs are higher than on the silver VMs, whose response times are higher than on bronze VMs. In contrast, JBQ dispatches requests to VMs according to their computational capacity, i.e., gold VMs process 4 times more requests than other VMs, and the average response times on all VMs are more balanced.

There are two observations about JBQ worth mentioning: The silver VM instance, S_8, appears to have more computational capacity than the other two silver VM instances, S_6, S_7, because it processes many more requests and yields lower response times than S_6 and S_7. JBQ is able to recognize and exploit such an opportunity. Furthermore, JBQ dispatches a very low number of requests to S_1, one of the bronze VMs. This observation leads us to think that JBQ is able to differentiate the VMs by their computational capacity and thus provides a convenient means to eliminate weak VMs in a cluster. In our future work, we will explore this property.

Table III: Comparison of different VMs under JBQ and bybusyness: the average request rate and average response times. s1 and s2 (bronze); s3, s4, and s5 (gold); s6, s7, and s8 (silver).

	$R_{byb}[ms]$	$\lambda_{byb}[reqs/s]$	$R_{JBQ}[ms]$	$\lambda_{JBQ}[reqs/s]$
s1	637.794	2.97	319.622	0.37
s2	767.969	2.60	360.122	1.66
s3	355.341	4.05	353.256	1.25
s4	341.553	4.04	478.202	8.47
s5	322.744	4.00	483.039	8.91
s6	408.325	4.16	383.519	2.12
s7	459.916	3.98	431.545	2.53
s8	385.376	4.17	434.017	4.65

D. A Cluster of Ten Silver VMs

We evaluate JBQ against bybusyness on a cluster of ten silver VMs, which may exhibit hidden heterogeneity, depending on the unknown neighboring effect. The total number of requests processed is 50000 and the arrival rate is 40 requests per second. We note that the difference among silver VMs' computational capacities is minor, compared to mixed instances, as indicated partially in Fig 3 (b). As such, we expect that the performance improvement of JBQ is minor, compared to bysusyness, depending on our estimation of computational capacities and overall workload in the cloud. We summarize our results in the silver cluster column of Table II. JBQ only improves the variance of response times and has a slightly worse performance in the average and higher percentile of response times. We attribute this observation to the prediction errors on the computational capacities. Overall, JBQ is able to achieve a comparable performance in a cluster of homogeneous instances, against the most prevailing and powerful load control scheme.

V. Related Work

We discuss the related work in two areas, one regarding the performance issues due to the heterogeneity in clouds and the other one regarding the load control algorithms.

While clouds are an emerging computing platform, various studies [9, 12, 20] present performance measurements of hosting different applications in clouds and report observations of high variability in the quality of service. Most of these studies particularly focus on the performance heterogeneity hidden within VMs of the same type, whereas [20] characterizes the cloud workload heterogeneity in terms of resource types (e.g., cores:RAM per machine) and their usage (e.g., duration and resources needed). Our study characterizes the heterogeneity of clouds systems from the perspective of the deployment of VMs and their performance in terms of computational capacities.

As a result, several studies [4, 5, 14, 25] on resource allocation and load control in the cloud try to explore the heterogeneity, mainly via simulation. Lee et. al [14] propose a scheduling scheme that allocates resources for an analytics cluster hosted in the cloud such that performance fairness is achieved. To minimize the operational total cost, Bjorkqvist et al. [4] develop an opportunistic replication policy, which tries to acquire VMs with more powerful computational capacities. A scheduler for a MapReduce application in heterogenous cloud environment is presented in [25]. The most closely related work is [5], which develops and implements a load control algorithm in Amazon EC2, with the objective of achieving similar response times on all VMs. However, in-depth profiling is required in that approach, whereas our proposed JBQ policy only needs to collect the number of oustanding requests and throughput.

There is a large body of related studies on load control, especially for homogeneous systems. For SIP clusters managing communication sessions of video and voice calls, Jiang et. al [11] propose an algorithm to forward the request to the server with the least amount of work assigned but not yet completed. Lu et. al [15] consider a homogeneous

cluster with multiple load controllers. To minimize the communication of the load controllers and the servers, they propose an algorithm, called Join-Idle-Queue. Exploring the locality and size of the requested data [8, 18, 21] is another important aspect to develop load controlling algorithms for homogeneous Web service systems. In contrast, our proposed JBQ algorithm aims to distribute the load on heterogenous systems, where the computational capacity of VMs is unknown and possibly fluctuates over time.

VI. Conclusion

In this paper, we study the heterogeneity of VMs deployed in clouds. Using the data collected from operational clouds as well as a deployed wikipedia system, we show that 35% of the deployed systems are composed of VMs of heterogeneous types and also that VM instances of the same configuration exhibit varying computational capacities. To address the high performance variability across heterogeneous VM instances, we develop a novel plug&play load-controlling algorithm JBQ, which dispatches requests to the VM with the shortest expected delay using an estimate of computational capacity. Relying on two simple statistics, i.e., the number of outstanding requests of a VM and the weighted average inter-departure times, collected in a non-intrusive manner, JBQ is able to effectively reduce the variance and higher percentile of response times across VMs, compared to the prevailing bybusyness policy in the Apache web server, for a wikipedia system deployed on heterogeneous VMs.

References

[1] D. Ansaloni, L. Y. Chen, E. Smirni, and W. Binder. Model-driven Consolidation of Java Workloads on Multicores. In *Proceedings of IEEE/IFIP DSN*, 2012.

[2] Apache JMeter. http://jmeter.apache.org/.

[3] Apache web server. http://httpd.apache.org/.

[4] M. Björkqvist, L. Y. Chen, and W. Binder. Opportunistic service provisioning in the cloud. In *Proceedings of IEEE Cloud*, pages 237–244, 2012.

[5] J. Dejun, G. Pierre, and C.-H. Chi. Resource provisioning of web applications in heterogeneous clouds. In *Proceedings of the 2nd USENIX conference on Web application development*, pages 5–5, 2011.

[6] A. EC2. http://www.amazon.com.

[7] Fedora. http://fedoraproject.org/.

[8] M. Harchol-Balter, M. Crovella, and C. D. Murta. On choosing a task assignment policy for a distributed server system. In *Computer Performance Evaluation (Tools)*, pages 231–242, 1998.

[9] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. J. Wasserman, and N. J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *CloudCom*, pages 159–168, 2010.

[10] K. R. Jackson, L. Ramakrishnan, K. J. Runge, and R. C. Thomas. Seeking supernovae in the clouds: a performance study. In *HPDC*, pages 421–429, 2010.

[11] H. Jiang, A. Iyengar, E. M. Nahum, W. Segmuller, A. N. Tantawi, and C. P. Wright. Design, implementation, and performance of a load balancer for sip server clusters. *IEEE/ACM Trans. Netw.*, 20(4):1190–1202, 2012.

[12] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *SIGMOD Conference*, pages 579–590, 2010.

[13] K. R. Krishna. Joining the right queue: A markov decision-rule. In *Proceedings of IEEE Conference on Decision and Control (CDC)*, 1987.

[14] G. Lee, B.-G. Chun, and R. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of Usenix HOTCLOUD*, 2011.

[15] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. R. Larus, and A. G. Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *Perform. Eval.*, 68(11):1056–1071, 2011.

[16] MediaWiki. http://www.mediawiki.org/.

[17] MySQL. http://www.mysql.com/.

[18] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of ASPLOS*, pages 205–216, 1998.

[19] PHP. http://www.php.net/.

[20] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of ACM Symposium on Cloud Computing (SOCC)*, pages 7:1–7:13, 2012.

[21] A. Riska, W. Sun, E. Smirni, and G. Ciardo. Adaptload: Effective balancing in custered web servers under transient load conditions. In *Proceedings of ICDCS*, pages 104–111, 2002.

[22] Y. Ueda and T. Nakatani. Performance variations of two open-source cloud platforms. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 1–10, 2010.

[23] A. Weinrig and S. Shenker. Greed is not enough: adaptive load sharing in large heterogeneous systems. In *INFOCOM*, pages 986 –994, mar 1988.

[24] Wikipedia. http://www.wikipedia.org/.

[25] H.-H. You, C.-C. Yang, and J.-L. Huang. A load-aware scheduler for MapReduce framework in heterogeneous cloud environments. In *Proceedings of SAC*, 2011.