Evaluating Weighted Round Robin Load Balancing for Cloud Web Services

Weikun Wang
Department of Computing
Imperial College London, UK
weikun.wang11@imperial.ac.uk

Giuliano Casale
Department of Computing
Imperial College London, UK
g.casale@imperial.ac.uk

***Abstract*—Weighted round robin load balancing is a common routing policy offered in cloud load balancers. However, there is a lack of effective mechanisms to decide the weights assigned to each server to achieve an overall optimal revenue of the system. In this paper, we first experimentally explore the relation between probabilistic routing and weighted round robin load balancing policies. From the experiment a similar behavior is found between these two policies, which makes it possible to assign the weights according to the routing probability estimated from queueing theoretic heuristic and optimization algorithms studied in the literature. We focus in particular on algorithms based on closed queueing networks for multi-class workloads, which can be used to describe application with service level agreements differentiated across users. We also compare the efficiency of queueing theoretic methods with simple heuristics that do not require to specify a stochastic model of the application. Results indicate that queueing theoretical algorithms yield significantly better results than than routings proportional to the VM capacity with respect to throughput maximization.**

I. Introduction

Load balancing services are increasingly offered by many cloud providers [1] [2]. Requests are dispatched by the load balancer to end servers following certain load balancing policies. These policies normally aim at minimizing the imbalance between different servers to improve system throughput or to reduce response time. Among commonly available load balancing polices, round robin is the most common one supported by major cloud providers [3]. Considering the heterogeneous resources available in the cloud, the weighted round robin policy, also offered by many cloud offerings, places more load to the servers with higher weight using a policy similar to round robin. This is much more suitable than round robin for cloud deployments to assign requests according to the capacity of the end servers. However, a challenging problem is how to decide the weights assigned to the servers. A simple intuitive way is to set the weights according to the computing power of the servers, but this ignores the fact that there could be different kinds of requests with different processing needs. These simple intuitive approaches do not guarantee an optimal performance for the system.

One alternative is to determine the weights from a probabilistic routing policy, which dispatches requests according to fixed probabilities. Queueing network models have been proposed by several research works as a way to determine the optimal probabilistic load balancing [4][5][6][7]. The work in [7], in particular, recently proposed an approximate solution with a guaranteed optimality ratio. However, this is determined under heavy-load assumptions for the optimization algorithm proposed therein and these assumptions may not be always applicable to real systems. Therefore, in this work we try to establish to what extent this heuristic is applicable in practice and the tradeoff incurred when using a more complex optimization-based solution at runtime, which is more difficult to implement, but does not require similar heavy-load assumptions.

In addition, the routing algorithm in [7] also supports multi-class workloads and presents the routing probability to each workload class. Traditional load balancing algorithms ignore the class of requests entering the system. However, this becomes a problem if different requests require different computing resources. This is also useful in real applications when the users have different privileges, e.g. golden, silver and bronze, which stand for different levels of services. Such levels of service can be formalised by means of service level agreements (SLAs).

In this paper, we start with the algorithms in [7] and experimentally explore the relation between probabilistic routing and weighted round robin load balancing for a web service deployed on Amazon EC2. From the experimental results a similar behaviour is found between the two kind of policies, which makes it possible to assign the weights of the weighted round robin policy according to the probability estimated from the heuristic algorithm and the optimization program introduced in [7] for probabilistic routing. Furthermore, an initial study has been done to compare these algorithms with an intuitive, empirical, weight assignment that weights each server according to its computational power. In this comparison, we define as revenue the total throughput of the system and we aim at finding a set of routing probabilities that can maximize it. We also explain how these algorithms can be used in practice, since some of the theoretical assumptions require slight modifications for application to the real systems. The results of our experiments suggest that the heuristic algorithm is able to provide a better revenue than näive assignments of weights. Yet, we show that further improvements can be achieved through runtime optimization using the algorithm in [7], at the expense of higher computational costs and implementation complexity, leading to the observation that heavy-load routing decisions are effective but not always optimal in lighter loads.

The rest of this paper is organized as follows. Section II gives background of load balancing in the cloud. Section III explains the load balancing algorithms for probabilistic routing. In Section IV, we evaluate different load balancing algorithms, and compare probabilistic routing and weighted round robin

TABLE I: Cloud load balancing policies

Cloud	*Type*	R	RR	WRR	LC	WLC	S
EC2 (ELB&Route53)	IaaS		*	*	*		
GoGrid	IaaS		*	*	*	*	*
Rackspace	IaaS	*	*	*	*	*	
App Engine	PaaS						*
Azure	PaaS		*				
CloudFoundry	PaaS		*	*	*		
Heroku	PaaS		*				

policies. Finally, Section V gives final remarks and future works.

II. Related work

Although research has fostered many works for a better understanding of the load balancing problem, simple policies are often offered to cloud users. We list here several popular load balancing policies: the Random (R) policy sends each request to a randomly selected server; the Round robin (RR) policy dispatches requests to each server in equal portions and in circular order; the Weighted round robin (WRR) policy assigns a fixed weight to each server; the Least connection (LC) policy connects the server with least active connections with clients; the Weighted least connection (WLC) policy assigns a weight to each end server while calculating the connections and the Source based (S) policy splits load based on source IP hashing. In [8], RR is shown to achieve good result for high load while returns poor performance under low to medium loads. LC, on the other hand, performs well for medium to high workloads, but it exhibits high waiting times for low load. All the weighted policies are more suitable for heterogeneous servers, but they face the problem of deciding the weights. Source based load balancing has been studied in [9] and shows good performance, however this approach has either high computational time or require additional monitoring effort.

Table I summarises the availability of these load balancing policies across some popular cloud offerings. In this table, we categorize the cloud providers by their service offering. The infrastructure as a service (IaaS) providers offer physical servers and virtual machines. The platform as a service (PaaS) providers deliver a computing platform for users to run their applications. It can be noticed that round robin and weighted round robin are the most popular policies in the current cloud offering.

In the next subsections we describe relevant research works that apply load balancing from the cloud provider or cloud user perspectives.

A. Provider-Level Load Balancing

The work in [1] proposes an offline optimization algorithm for SLA-aware geographical load balancing in data centers with an online approach to account for the uncertainty of the electricity prices. Experiment against a heuristic greedy method shows significant cost savings for the proposed approach. In [2] the authors present a geographical load balancer among geographically distributed data centers. An offline deterministic optimization method for design time usage and an online VM placement, migration and geographical load balancing algorithm for runtime are demonstrated based on the predictions on workload, energy prices, and renewable energy generation capacities. Experiment against simulation data shows that the online algorithm has worse performance than the offline one due to incomplete information. On the other hand, the online algorithm has a strong impact on the quality of the solutions. [10] proposes an online load balancing algorithm considering VM resource heterogeneity. The load balancer uses the number of outstanding requests and the inter-departure times in each VM to dispatch requests to the VM with the shortest expected response time. Experiment against a common policy in the Apache web server shows that the proposed algorithm is able to improve variance and achieve higher percentile of response time. In [11], the authors propose a self-organizing approach to provide robust and scalable solutions for service deployment, resource provisioning, and load balancing in a cloud infrastructure. The algorithm proposed has the additional benefit to leverage cloud elasticity to allocate and deallocate resources to help services to agree with the contractual SLAs. [12] shows a cost minimization mechanism for data intensive service provision. The proposed mechanism uses a multi-objective genetic algorithm to manage data application services and to produce optimal composition and load balancing solutions.

B. User-Level Load Balancing

We now consider the case where load balancing is implemented by the cloud user. [13] proposes a dynamic programming approach to find an optimal distribution of the query workload for hybrid clouds. The authors implement the solution over an Hadoop/Hive cloud based infrastructure and the experiments demonstrate a major performance improvement in full compliance with cloud usage constraints. The work in [14] focuses on multiple IaaS service centers. A non-linear model for capacity allocation and load redirection of multiple request classes is proposed and solved by decomposition. A comparison against a set of heuristics from the literature shows that the proposed algorithm overcomes heuristic approaches without penalizing SLAs and it is able to produce results that are close to the global optimum. The authors in [15] formulate an optimization problem faced by a cloud procurement endpoint (a module responsible for provisioning resources from public cloud providers), where heavy workloads are tackled by relying on the public clouds. They develop a linear integer program to minimize the resource cost and evaluate how the solution scales with different problem parameters such as task deadline. In [16] a structured peer-to-peer network, based on distributed hash tables, is proposed to support service discovery, self-managing, and load-balancing of cloud applications. The effectiveness of the peer-to-peer approach is demonstrated through a set of experiments on Amazon EC2. Finally, [17] proposes an adaptive approach for component replication of cloud applications, aiming at cost-effective placement and load balancing. The distributed method is based on an economic multi-agent model and is able to achieve high application availability guaranteeing at the same time service availability under failures.

III. Techniques

The load balancing system assumed in this paper contains a load balancer and several target servers. The users send requests to the load balancer, which then dispatches the requests

to the target servers following a static policy. The target servers are assumed to be arranged in a parallel topology, which means that once the server finishes processing the requests, it will return the result directly to the user. Another assumption is that there is a fixed number of jobs running in the system, which represents typical admission control (i.e., concurrency limits) in servers.

The system resembles a closed queueing network. The jobs in this closed network are requests from the users to the target servers and think time represents the waiting time between successive jobs. The performance of the load balancing system is here evaluated by a revenue, which can be defined in many ways, e.g. the mean throughput or the mean response time of the requests. Here we use the throughput of the system since we are assuming a closed topology and therefore throughput maximization and response time minimization are equivalent. We define M to be the number of servers and R to be the number of users classes. The notion of class may be defined in different ways, we here interpret it as a set of requests that requires a specific performance. This is the case, for instance, of systems with differentiated SLAs, where users can be of different classes (e.g., gold, silver, bronze) and each class experiences different quality of service from the server. The revenue Γ for each server is defined as the weighted sum of the throughputs of each class of users, which could be defined as:

$$\Gamma = \sum_{r=1}^{R} w_r X_r \tag{1}$$

where w_r is the revenue weight for users of class r and X_r is the corresponding mean throughput. We similarly define X_{ir} to be the throughput of the users of class r at VM i. Our model assumes that VM operate in parallel, therefore $X_r = \sum_{i=1}^{M} X_{ir}$.

The work in [7] sets out to maximize the revenue (1) and and it does so by setting up a closed multi-class queueing model to describe the dependence between X_r, the capacity of the servers, and the number of concurrently executing requests. In the queueing model, the servers are treated as queueing stations with a delay node representing the load balancer to send different classes of requests to the servers. The routing probability p_{ir} indicates the probability of the request of class r to be be dispatched to the ith server from the delay node.

In [7], an optimization program (OPT) is proposed to maximize the revenue (1). We define H_r to be the set of servers that requests of class r are allowed to visit, N_r to be the number of users for jobs of class r, μ_{ir} to be the mean service rate for jobs of class r at server i. X is the matrix of X_{ir} values and L is the vector of L_r values. X and L are assumed to be real numbers. OPT sets out to solve the following optimization problem:

$$\begin{aligned} OPT: \max_{X \in R_+^{MR}, L \in R_+^{M}} & \sum_r w_r \sum_{i \in H_r} X_{ir} \\ s.t.: & \sum_{i \in H_r} \frac{X_{ir}}{\mu_{ir}} L_i = N_r, \forall r \\ & \sum_{r: i \in H_r} \frac{X_{ir}}{\mu_{ir}} = 1, \forall r \end{aligned} \tag{2}$$

Algorithm 1 Heuristic algorithm

Require: μ_{ir}, w_r, H_r, m
$r^*(i) = \arg\max_{r: i \in H_r} w_r \mu_{mr}$
$p_{ir^*(i)} = \frac{\mu_{ir^*(i)}}{\sum_{j=m,\ldots,M:\ r^*(j)=r^*(i)} \mu_{jr^*(j)}}$, $\forall i = m, \ldots, M$
$p_{ir} = 0, \forall i \neq r^*(i), i = m, \ldots, M$
Distribute p_{ir} equally on all the server i, $i = 1, \ldots, m-1$ so that $\sum_{i \in H_r} p_{ir} = 1$
return $\boldsymbol{p}$

It can be noticed that the objective function in (2) aims at maximizing the revenue, which is defined in (1), observed at the output of all the parallel servers of the network. The optimal throughput X_{ir}^* and queue length L_{ir}^* obtained from the optimization problem are defined by a set of routing probabilities p_{ir}^*. Considering the load balancing system has a parallel topology, the visit ratio $v_{ir} = p_{ir}$. Therefore according the forced flow law [24], we have $p_{ir}^* = \frac{X_{ir}^*}{\sum_{j \in H_r} X_{jr}^*}$ for all VMs i and user classes r.

In [7] the authors have shown that as the system becomes larger, it becomes increasingly difficult for OPT to find a local optimum. Moreover, it is computational expensive and requires the number of requests N_r. To cope with issue, another algorithm introduced in [7] is a revenue maximization algorithm for heavy workload, which is a heuristic solution to OPT introduced above. The heuristic algorithm finds the optimal routing probability p_{ir} from the load balancer to the servers given the service rate μ_{ir} for each class of the request at each server, where $i = 1, \ldots, M$, $r = 1, \ldots, R$. The pseudocode can be found in Algorithm 1. In the pseudocode, it is assumed that the servers are indexed in a way that

$$\max_{r: 1 \in H_r} w_r \mu_{1r} \leq \max_{r: 2 \in H_r} w_r \mu_{2r} \leq \cdots \leq \max_{r: M \in H_r} w_r \mu_{Mr}$$

The advantage of the heuristic algorithm is that it is independent of the number of requests N_r for each class and it has been proved to achieve an optimality ratio of $1+1/(M-1)$ under heavy load, thus it gets closer to optimality with increasing number of VMs. It is also computational efficient. However, this method is developed under the assumption of heavy traffic. Although in the original paper the authors have shown that the algorithm also obtains good result for normal load, it still needs to be validated in real applications. The service rate μ_{ir} can be obtained from the service demand D_{ir}, which we approximate with the CI algorithm introduced in [18]. Then μ_{ir} can be obtained by setting $\mu_{ir} = \frac{p_{ir}}{D_{ir}}$.

Therefore we can see that the heuristic algorithm and OPT program are complementary to each other since the former performs well for small systems and the latter is developed under heavy load assumptions and better suited for large systems. Considering that OPT does not scale to larger systems as show in [7], whereas the heuristic algorithm does scale, in this paper we limit the comparison to a small sized deployment where OPT and the heuristic can be compared.

The two approaches proposed in [7] are for probabilistic routing. However, as we have already shown in Section II that many cloud providers support weighted round robin. Therefore

it will be meaningful to define the weights based on the probabilities. Since the weights of the servers have to be integers normally, we can obtain weights by $W = [p * 100]$.

IV. Experimental evaluation

In this section, we discuss the experiments for the comparison between probabilistic routing and weighted round robin policy and the performance of different weights assigning algorithms. For ease of analysis of the experimental results, we set the weight of the revenue $w_r, r = 1, \ldots, R$ to be 1 for all the requests. In the following, the testbed description is first described. Then we discuss some issues that arise in the implementation of the above policies. Finally, the results of the experiment are illustrated and discussed.

A. Testbed Description

To assess the performance of different load balancing algorithms, we setup an environment with Apache Open for Business (OFBiz) application [19]. The application is deployed on Amazon EC2. OFBiz is an open source enterprise resource planning system. The web server used is the embedded Tomcat inside OFBiz and for the database we use the default Derby database. Therefore the database is deployed with each instance. The user requests are generated from OFBench [20], a load generator for OFBiz. We have customised OFBench to run only two types of sessions in order to simplify the analysis. Each session represents a different type of users and some requests may belong to either session. We define custom user sessions in OFBench to have different classes of users. The load balancer used is Haproxy v1.5 [21], which is a popular TCP/HTTP load balancer that supports weighted round robin. The pseudocode for weighted round robin implemented in the Linux Virtual Server [22] is shown in Algorithm 2. In the algorithm, $\boldsymbol{W} = (W_0, \ldots, W_{M-1})$ is the weight vector, where i is the server selected during last iteration, W_i is the weight for server $i+1$ with $i = 0, \ldots, M-1$ and cw is the current weight. gcd is the greatest common divisor. The actual implementation in Haproxy is similar but more complicated considering also the current number of running threads inside the backend servers. But they both allocate jobs proportional to the weights for each server. To monitor the performance behaviour of each VM and OFBiz, we use the MODAClouds Monitoring Platform [23]. A Data Collector is deployed on each VM and it is able to monitor the CPU utilization of each VM and continuously parse the OFBiz logs to get the response time and arrival time of the requests at the server. Then the Data Collector will send the monitoring data to the Deterministic Data Analyzer. The Deterministic Data Analzyer processes at high-speed the monitoring data coming from the Data Collectors with operations like computing average or maximum values during a time window. Finally, the Deterministic Data Analyzer will send the processed monitoring data to an Observer that archives this data. We use this archived data to evaluate the outcome of the load balancing experiments. The interaction between each component is shown in Figure 1. The default Derby database is co-located with the application server. This is more appropriate for an initial evaluation of the load balancing policies in [7] since otherwise the system will not be a parallel topology.

Algorithm 2 Weighted round robin

Require: W
  $i = -1$
  $cw = 0$
  while TRUE **do**
    $i = (i + 1) \bmod n$
    if $i == 0$ **then**
      $cw = cw - gcd(\boldsymbol{W})$
      if $cw < 0$ **then**
        $cw = \max(\boldsymbol{W})$
        if $cw == 0$ **then**
          return null
        end if
      end if
    end if
    if $W_i \geq cw$ **then**
      return i
    end if
  end while

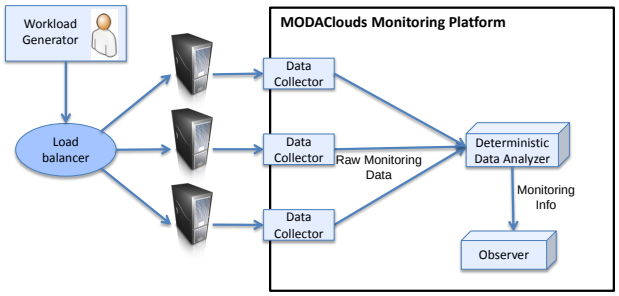


Fig. 1: Load balancing system

We have deployed each component of the system on Amazon EC2. The configuration of different instances we use is shown in Table II. Each experiment lasts for one hour. The number of clients, the think time as well as the workload differ for each experiment. All the instances are in the Ireland EU-WEST-1A region.

B. OFBiz Configuration and Workload Definition

During our evaluation of the algorithms, we notice some experimental issues that needed to be addressed to run the experimental comparison. The most challenging problem is how to adapt the algorithms to multi-core instances.

1) Multi-core Adaptation: Both the heuristic and optimization algorithms are defined using queueing models with single-server queues. This modeling abstraction corresponds to assuming that servers have a single core. However, nowadays servers and VMs normally have multiple cores. In order to adapt the algorithms for multi-core instances, we propose two methods. One is to divide the demand by the number of cores while applying the algorithms. In particular, we use the demand

$$D^*_{ir} = D_{ir}/c_i \tag{3}$$

where c_i is the number of cores for server i. Another approach is to assume the multi-core machine is represented by several single core machines while computing the routing probability. Then we sum the probabilities of the one core machines to get

TABLE II: Instance types

VM Type	ECU	vCPU	Memory
m1.medium	2	1	3.7 GB
c1.medium	5	2	1.7 GB
c3.large	7	2	3.7 GB
c3.xlarge	14	4	7.5 GB

TABLE III: Request setup for 2 classes of users

Request	Session 1	Session 2
checkLogin	2	2
login	1	1
logout	1	1
main	2	3
quickadd		1

the original routing probability. We set demands to be

$$D^*_{isr} = D_{ir}, \quad s = 1, \ldots, c_i. \tag{4}$$

This is assuming the demand for each core is the same. After obtaining the new routing probability p^*_{isr} from the algorithms, we get the original p_{ir} by

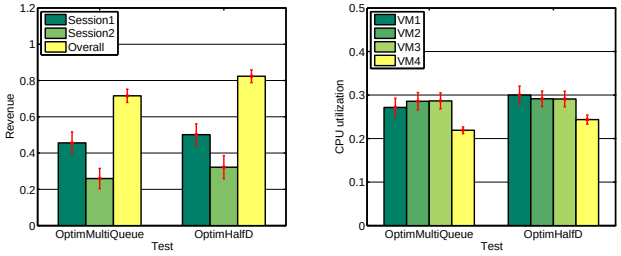
$$p_{ir} = \sum_{s=1}^{c_i} p^*_{isr} \tag{5}$$

The formula means the final routing probability p_{ir} comes from the sum of the routing probability for each core p_{isr}.

To compare the two approaches defined above, we run an experiment. The experiment setup is shown in Table IV. We use the probabilistic routing policy with routing probability of 20% for VM1, VM2, VM3 and 40% for VM4. For simplicity, we call the two proposed methods *OptimHalfD* to refer to the method based on (3) and *OptimMultiQueue* to refer to the method based on (4)-(5). The number of user classes is 2, which is listed in Table III and the number of users $N_r = 12$ for each of the user. The think time is set to 4 seconds between each request and session. The revenue Γ and CPU utilization is shown in Figure 2 together with the confidence intervals. From the figures we can find that taking half demand of the 2-core instance has better revenue than the other approach. Moreover, it has more balanced CPU utilization. Therefore in the following experiments when we use a multi-core server, we will scale the demand according to (3) while applying the heuristic and optimization algorithms.

C. Experimental Result

In this subsection, we study the relations between probabilistic routing (PR) and weighted round robin (WRR) policies and introduce the result of the algorithms under different number of users classes. The detailed setup is shown in Table IV. We mainly test four different load balancing policies: weights 1,1,1,2 respectively for VM1 to VM4 which are proportional to the number of cores of the instances; weights 1,1,1,3.5 respectively for VM1 to VM4 which are proportional to the number of ECUs (which a standard Amazon measure of the processing power of an instance) of the instances; weights returned from the heuristic algorithm and weights returned from the optimization program. For simplicity we identify the four cases as: *1112*, *1113.5*, *Heur* and *Optim*.



(a) Revenue (b) CPU utilization

Fig. 2: Revenue and CPU utilization

TABLE IV: Experiment setup

VM	No. CPU	Location
DDA	1	m1.medium
Observer	1	m1.medium
Haproxy	1	m1.medium
VM1 (OFBiz)	1	m1.medium
VM2 (OFBiz)	1	m1.medium
VM3 (OFBiz)	1	m1.medium
VM4 (OFBiz)	2	c3.large
OFBench1	4	c3.xlarge
OFBench2	4	c3.xlarge
OFBench3	4	c3.xlarge
OFBench4	4	c3.xlarge

Both the heuristic algorithm and the optimization algorithm take the demands from the *1112* case with weighted round robin policy since this policy is reasonably fair and balanced for all the machines and the demands are correct. For the heuristic algorithm, we follow the recommendations in [7] and set $m = 2$ in Algorithm 1, a setting which guarantees the existences of a solution.

1) Comparison Between PR and WRR: Here, we study the relation between PR and WRR. Two different classes of users are used. The detailed composition of the sessions set in OFBench is shown in Table III. The number of users N_r is set to 40 for each user and the think time between requests and sessions is set to 2 seconds. Each experiment lasts one hour. The probabilities for both algorithms are shown in Table V and VI. For the WRR policy, since Haproxy does not support fractional numbers to be used as weights, we round the ratio of the probabilities returned from heuristic and optimization algorithms to integers. Since VM1, VM2 and VM3 are all installed in a *m1.medium* instance and we found similar demands for all the requests, we average the demands taken from the three VMs so that two OFBiz running on the same *m1.medium* instance have identical demands. This is because small differences in the demands can result in macroscopic differences in the load balancing decisions of the heuristic in particular. However if the VMs have the same instance class but with different hardware, the demands would be quite different even if the instance class is the same.

Discussion. The revenue, CPU utilization and the response time of the sessions are shown in Figure 3, 4 and 5. From the figures, we can see that the PR policy has a trend similar to WRR. In particular, regarding the revenue, it can be noticed that WRR has larger revenue than the PR policy for all the tests. A larger total throughput also contributes to higher CPU utilization for the WRR case than PR according to Little's law [24]. We think the reason behind this is that WRR is

TABLE V: Heur - 2 classes

VM	PR		WRR	
	Session 1	**Session 2**	**Session 1**	**Session 2**
VM1	0	1	0	1
VM2	0.1651	0	0.2143	0
VM3	0.1651	0	0.2143	0
VM4	0.6699	0	0.5714	0

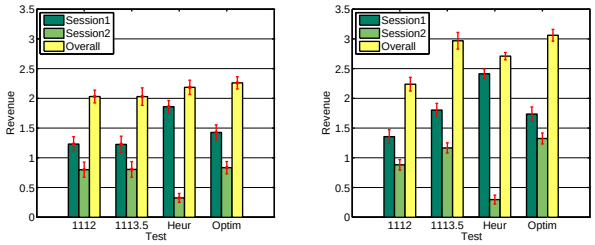
TABLE VI: Optim - 2 classes

VM	PR		WRR	
	Session 1	**Session 2**	**Session 1**	**Session 2**
VM1	0.2843	0	0.3333	0.0133
VM2	0.2843	0	0.3333	0.0133
VM3	0.2843	0	0.3333	0.0133
VM4	0.1499	1	0	0.9600

a more balanced and stable policy for distributing requests while PR relies on a probability distribution which may incur oscillating load on the instances. Overall, the optimization algorithm returns the best revenue for the PR and WRR policies. The heuristic also returns good results compared to other tests and its response time is quite balanced across servers. We also show the per minute average response time and the number of outstanding requests for the OPT policy in Figure 6 and 7. From the figure we can see that the response time remains stable and for each instance the response time is very similar. PR shows a small peak in the response time due to a peak of requests in Figure 6a. In summary, the results suggest that the probability returned by the probabilistic algorithms is well suited to assign the weights of WRR.

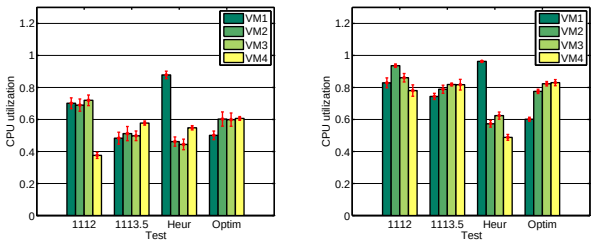
2) 4 Classes of Requests: Here, we study the load balancing for 4 classes of users. The detailed composition of the sessions set in OFBench is shown in Table VII. The load to the server increases from users of Session 1 to Session 4. The number of users $N_r = 9$ for each user and the think time between requests and sessions is set to 2 seconds. We changed the number of users compared to the 2 classes of requests case because we have more user classes, which makes the total number of users increase. To make sure the server is not overloaded, we decrease the number of users for each class. Since in the previous experiment we show that the probabilities from heuristic and optimization algorithms could be directly applied to the weighted round robin policy for setting the weights, here we will only show the result of the WRR policy. The probabilities of the two algorithms are demonstrated in Table VIII and IX.

Discussion. The revenue, CPU utilization and the 95th percentile of the response time of the sessions are shown in Figure 8 and 9. For the 4 classes case, the observation is mostly consistent with that of the 2 classes case. The optimization program returns the best revenue among all the policies. and the revenue is around 27% higher than others. Besides, the optimization algorithm has a more balanced response time compared to the heuristic because it assigns all the users of session 2, 3, 4 to VM1 which makes the response time higher for requests served by that VM instance. For the demand estimated using the CI algorithm, we can see that VM4 has larger demand than other instances which we believe is because VM4 has the heaviest load of the 4 VMs.



(a) PR (b) WRR

Fig. 3: Revenue - 2 classes



(a) PR (b) WRR

Fig. 4: CPU utilization - 2 classes

D. Discussion

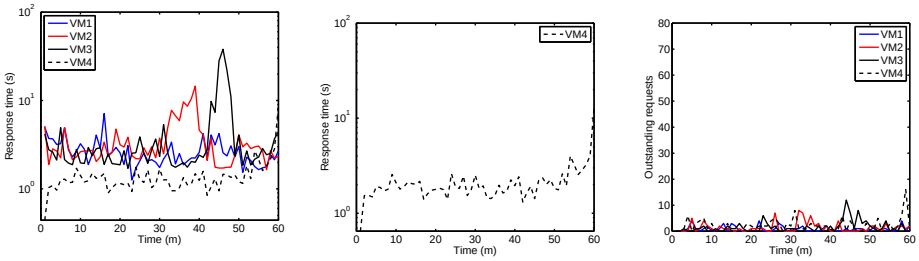
The observations through the experimental study may be summarized as the following points:

- PR returns similar results as the WRR policy
- Setting weights of WRR from the probability of PR is possible
- OPT returns the best revenue
- Heuristic gives better result than simple weight setting

For different weights assigning algorithms, OPT is computationally expensive and requires more parameters than the heuristic approach, such as the total number of users. It returns the best system revenue compared with all the other approaches shown in the experiment. The heuristic algorithm requires only the demands but the problem is that it only works best for heavy load. For the simple weights setting approaches, they require the minimum effort to decide the weight, e.g. a value proportional to the number of cores and their computation power. However, they do not provide good system revenue compared to algorithms that use queueing theoretic models.

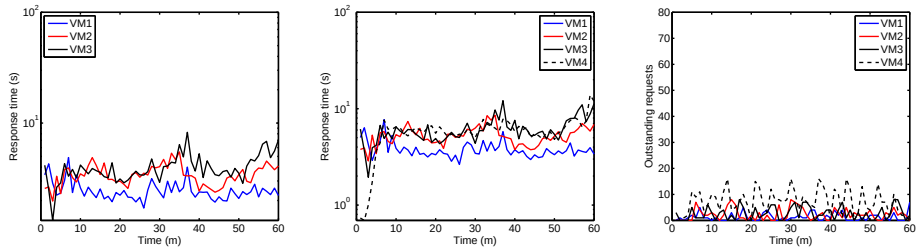
V. Conclusion and Future Work

The weighted round robin is a load balancing policy offered by many popular cloud providers. However, there is a lack of effective mechanisms to decide the weight assigned to each server to achieve an overall optimal revenue of the system. In this paper, we experimentally explore the relation between probabilistic routing and weighted round robin policies. From the experiment a similar behavior is found between these two, which makes it possible to assign the weights according to the routing probability estimated from queueing theoretic heuristic and optimization algorithms in the literature. The algorithms described also support multi-class workload which could be



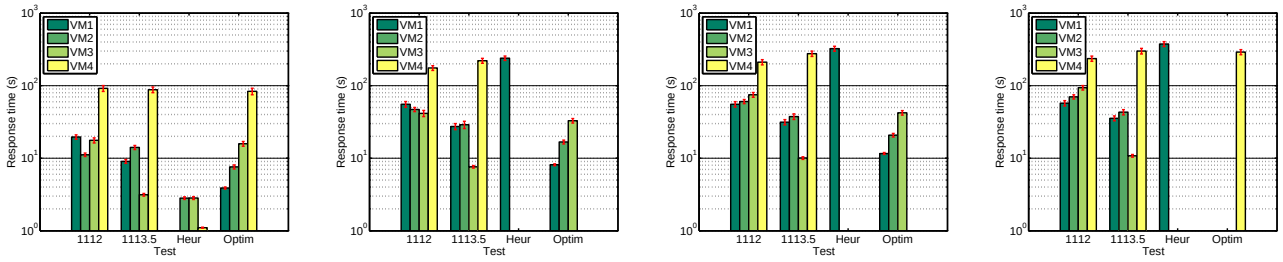
(a) Response time - Session 1 (b) Response time - Session 2 (c) Outstanding requests

Fig. 6: Response time and outstanding requests - PR - 2 classes



(a) Response time - Session 1 (b) Response time - Session 2 (c) Outstanding requests

Fig. 7: Response time and outstanding requests - WRR - 2 classes



(a) Response time - Session 1 (b) Response time - Session 2 (c) Response time - Session 3 (d) Response time - Session 4

Fig. 9: Response time (95th percentile) - 4 classes

TABLE VII: Request setup for 4 classes of users

Request	Session 1	Session 2	Session 3	Session 4
addtocartbulk		1	1	1
checkLogin	2	2	2	2
checkoutoptions			3	4
login	1	1	1	1
logout	1	1	1	1
main	2	3	3	3
orderhistory				1
quickadd		1	1	1

TABLE VIII: Heuristic method - 4 classes

VM	Session 1	Session 2	Session 3	Session 4
VM1	0	1	1	1
VM2	0.3300	0	0	0
VM3	0.3485	0	0	0
VM4	0.3215	0	0	0

applied to applications with service level agreements differentiated across users. Comparison against simple heuristics that do not require to define stochastic models, but simply sends load proportionally to the compute capacity, of the application indicates that the optimization algorithm is able to provide good routing decisions to optimize revenue.

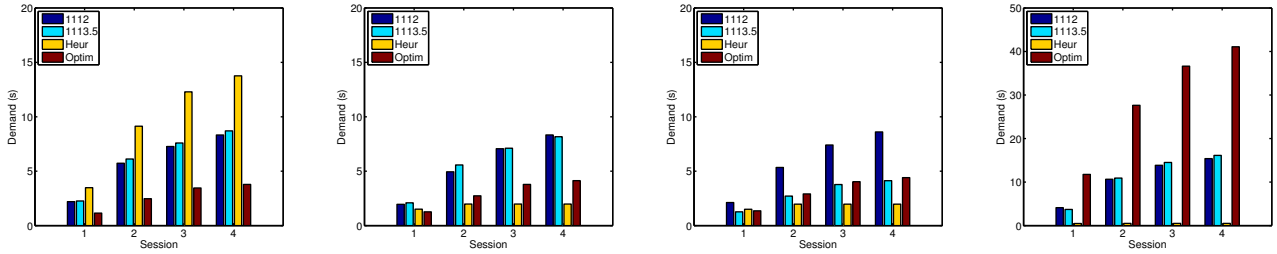
TABLE IX: Optimization method - 4 classes

VM	Session 1	Session 2	Session 3	Session 4
VM1	0.3154	0.3333	0.3333	0
VM2	0.3154	0.3333	0.3333	0
VM3	0.3154	0.3333	0.3333	0
VM4	0.0539	0	0	1

In the future, we are planing to use OPT and the heuristic as part of a self-adapting load balancing mechanism. We plan to change the routing decision adaptively at run time and possibly apply a load-dependent algorithm to compare with the optimization and heuristic algorithms. In addition, it would be interesting to perform a systematic comparison between the optimization algorithm based weighted round robin policy and other policies such as the LC and S policies.

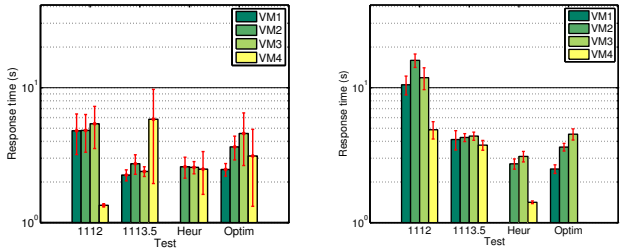
ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement n 318484 and from an AWS in Education research grant. The authors wish to thank Karsten Molka for the help with the scripts for the plot.



(a) VM1 (b) VM2 (c) VM3 (d) VM4

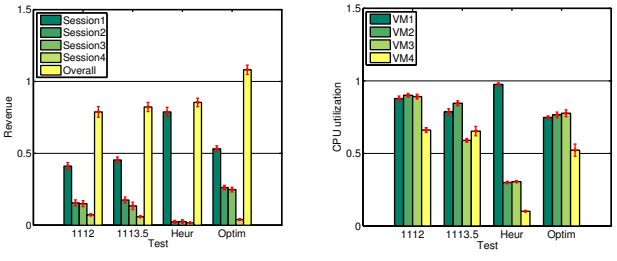
Fig. 10: Demands - 4 classes



(a) Session 1 - PR (b) Session 1 - WRR

(c) Session 2 - PR (d) Session 2 - WRR

Fig. 5: Response time - 2 classes



(a) Revenue (b) CPU

Fig. 8: Revenue and CPU utilization - 4 classes

REFERENCES

[1] M. A. Adnan, R. Sugihara, and R. K. Gupta, "Energy efficient geographical load balancing via dynamic deferral of workload," in *Proc. of IEEE Cloud*. 188-195, 2012.

[2] H. Goudarzi and M. Pedram, "Geographical load balancing for online service applications in distributed datacenters," in *Proc. of IEEE Cloud*. 351-358, 2013.

[3] K. Boloor, R. Chirkova, T. Salo, and Y. Viniotis, "Heuristic-based request scheduling subject to a percentile response time sla in a distributed cloud," in *Proc. of IEEE GLOBECOM*. 1–6, 2010.

[4] H. Kobayashi and M. Gerla, "Optimal routing in closed queuing networks," *ACM TOCS*, 1(4), 294-310, 1983.

[5] W. C. Cheng and R. R. Muntz, "Optimal routing for closed queueing networks," *Perform. Eval.*, 13(1), 3-15, 1991.

[6] A. Hordijk and J. Loeve, "Optimal static customer routing in a closed queuing network," *Statistica Neerlandica*, 54(2), 148-159, 2000.

[7] J. Anselmi and G. Casale, "Heavy-traffic revenue maximization in parallel multiclass queues," *Perform. Eval.*, 70(10), 806-821, 2013.

[8] Y. M. Teo and R. Ayani, "Comparison of load balancing strategies on cluster-based web servers," *Simulation*, 77(5), 185-195, 2001.

[9] Z. Cao, Z. Wang, and E. Zegura, "Performance of hashing-based schemes for internet load balancing," in *Proc. of IEEE INFOCOM* , 332-341, 2000.

[10] S. Spicuglia, L. Y. Chen, and W. Binder, "Join the best queue: Reducing performance variability in heterogeneous systems," in *Proc. of IEEE Cloud*. 139-146. 2013

[11] B. A. Caprarescu, N. M. Calcavecchia, E. Di Nitto, and D. J. Dubois, "Sos cloud: self-organizing services in the cloud," in *Bio-Inspired Models of Network, Information, and Computing Systems*. 48-55. 2012.

[12] L. Wang and J. Shen, "Towards bio-inspired cost minimisation for data-intensive service provision," in *Services Economics (SE), 2012 IEEE First International Conference on*. 16-23. 2012.

[13] K. Y. Oktay, V. Khadilkar, B. Hore, M. Kantarcioglu, S. Mehrotra, and B. Thuraisingham, "Risk-aware workload distribution in hybrid clouds," in *Proc. of IEEE CLoud*. 229-236. 2012.

[14] D. Ardagna, S. Casolari, M. Colajanni, and B. Panicucci, "Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems," *Elsevier JPDC*, 72(6). 796-808, 2012.

[15] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Proc. of IEEE CLoud*. 228-235. 2010.

[16] R. Ranjan, L. Zhao, X. Wu, A. Liu, A. Quiroz, and M. Parashar, "Peer-to-peer cloud provisioning: Service discovery and load-balancing," in *Cloud Computing*. Springer, 195-217. 2010.

[17] N. Bonvin, T. G. Papaioannou, and K. Aberer, "An economic approach for scalable and highly-available distributed applications," in *Proc. of IEEE CLoud*. 498-505. 2010.

[18] J. F. Pérez, S. Pacheco-Sanchez, and G. Casale, "An offline demand estimation method for multi-threaded applications," in *Proc. of IEEE MASCOTS*. 21-30. 2013.

[19] http://ofbiz.apache.org/.

[20] J. Moschetta and G. Casale, "Ofbench: an enterprise application benchmark for cloud resource management studies," in *Proc. of IEEE MASCOTS*. 393-399. 2012.

[21] http://www.haproxy.org/.

[22] http://kb.linuxvirtualserver.org/wiki/Weighted_Round-Robin_Scheduling.

[23] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu *et al.*, "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds," in *Proc. of IEEE MISE*. 50-56. 2012.

[24] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.