Resource Allocation and Management in Cloud Computing

Amir Nahir
Department of Computer Science
Technion, Israel Institute of Technology
Haifa 32000, Israel
Email: nahira@cs.technion.ac.il

Ariel Orda
Department of Electrical Engineering
Technion, Israel Institute of Technology
Haifa 32000, Israel
Email: ariel@ee.technion.ac.il

Danny Raz
Department of Computer Science
Technion, Israel Institute of Technology
Haifa 32000, Israel
Email: danny@cs.technion.ac.il

***Abstract*—Resource allocation and management in Cloud Computing is a very complex task. This is mainly due to the scale of the cloud and the number of services deployed in it. Since cloud users and service providers are given access to supercomputer-level resources, their effect over the cloud's overall performance is greater than ever. This raises multiple research questions related to the management and performance of cloud computing systems in light of the end-users selfishness. In this work we specifically study the overall performance when selfish service providers may split work between the (shared) cloud and private resources. The size of modern data center and the number of service housed in it calls for fully distributed management solutions. We propose task assignment policies that are specifically adequate for large-scale distributed systems, and show that they provide new capabilities in improving system performance. In particular, we develop new resource allocation algorithms that converge to a working point that balances the end-user experience with the operational costs of leasing resources from the cloud provider.**

I. Introduction

Cloud computing is a new and emerging paradigm in which jobs are processed by several services deployed on a set of distributed servers and accessed via a common network. This network of servers and devices, collectively known as *the cloud*, can offer a significant reduction in computing and storage costs due to economy of scale. In fact, computing at the scale of the cloud allows users to reach into the cloud for resources as they need them, and to attain supercomputer-level power using any standard client with network connectivity.

The Cloud Computing paradigm is based on three fundamental types of entities: the cloud provider, the service provider and the end user. The cloud provider owns and manages the infrastructure, constructed in form of large scale data centers. The service provider deploys its service in the data center, and the end user accesses it to have its jobs processed. Many end users may access a single service, and many services can be deployed in the same data center.

The scale of data centers, coupled with the self-optimizing nature of the different services deployed in them, prohibits central coordination and brings about many challenges in the realm of resource allocation and management. In this thesis [13] we investigate several problems in this domain[1].

We consider the case where service providers (SPs) may split their incoming traffic between privately-owned resources and the cloud (this approach is termed *workload factoring* [7], [24]). Unlike the private resource, which provides guaranteed performance, the performance of the shared resource is highly dependent on the usage pattern of other users (SPs), which in turn influences a user's decision if and to what extent to make use of the shared resource. The intrinsic relation between the utility that a user perceives from the shared resource and the usage pattern followed by other users gives rise to a non-cooperative game, which we model and investigate.

With the growth in adoption of the Cloud Computing paradigm, more and more SPs choose to completely give up keeping any significant private computational resource. In such a setup, SPs no longer split their job between a private and a shared resource they send the entire job for processing in the cloud. Consequently, the service's performance is completely dependent on the performance of the cloud. Once a job is sent to the cloud, the scheduler's assignment of the user's job to the specific processing server (resource) may have crucial implications on the quality of the service provided to the user. Furthermore, existing schedulers often incur a high communication overhead when collecting the data required to make scheduling decisions, hence delaying job requests on their way to the processing servers.

We propose a novel scheme that incurs no communication overhead between the users and the servers upon job arrival, thus removing any scheduling overhead from the job's critical path. Our approach is based on creating several replicas of each job and sending each replica to a different server. Upon the arrival of a replica to the head of the queue at its server, the latter signals the servers holding replicas of that job, so as to remove them from their queues. We show, through analysis and simulations, that this scheme significantly improves the expected queuing overhead over traditional schemes under various load conditions and different job length distributions. In addition, we show that our scheme remains efficient even when the inter-server signal propagation delay is significant (relative to the jobs execution time). We provide a heuristic solution to the performance degradation that occurs in such cases and show, by simulations, that it efficiently mitigates the detrimental effect of propagation delays.

A key feature that makes cloud computing attractive to service providers is *elasticity*, i.e., the ability to dynamically change the amount of allocated resources. This is typically done by adjusting the number of virtual machines (VMs) running a service based on the current demand for that service.

[1]The results of our research have been published in [10], [11], [14], [16]–[18]

978-3-901882-76-0 @2015 IFIP

For large scale services, centralized management is impractical and distributed methods are required. In such settings, no single component has full information on demand and service quality, thus elasticity becomes a real challenge. We address this challenge by proposing a novel elasticity scheme that enables fully distributed management of large cloud services. Our scheme is based on two main components, namely, a task assignment policy and a VM management policy. The task assignment policy strives to "pack" VMs while maintaining SLA requirements. The VM management policy is based on local activation of new VMs and self-deactivation of VMs that are idle for some duration of time. Through analysis, simulations and an implementation, we demonstrate that our scheme quickly adapts to changes in job arrival rates and minimizes the number of active VMs so as to reduce the operational costs of the service, while adhering to strict SLA requirements

A. Research Questions

This thesis [13] addresses issues in the realm of large scale distributed Cloud-based services. Specifically:

1) How would service providers partition their load between a privately owned resource and the cloud in a non-cooperative setup?
2) Can the management overhead of data center task schedulers be reduced to allow scalability?
3) Can a scalable distributed cloud elasticity scheme balance service performance with operational costs?

B. Contribution

The main contributions of our research are as follows.

1) Formal modeling and analysis of the Workload Factoring problem as a noncooperative game. We establish the existence and uniqueness of the game's Nash equilibrium, analyze the game's Price of Anarchy, and investigate some practical implications of our formal model.
2) A novel, replication-based, load balancing scheme. We analyze the proposed scheme for the case of Poisson distributed job arrivals and exponentially distributed job lengths, and conduct a comprehensive study comparing the proposed scheme to the Supermarket Model under several different load patterns. We further propose a heuristic that assists in improving the performance of replication-based load balancing in face of moderate signal propagation delay.
3) A fully distributed scheme for elasticity of large scale cloud-based services, decoupling the task assignment policy from the VM management policy. We evaluate the proposed scheme using a variety of load patterns. Our evaluation is based on simulations and an implementation, the latter in the framework of Amazon's EC2 (emulating a real-life commercial environment).

An additional contribution of this thesis, published in [10], [14], addresses the structure of networks arising in a noncooperative setup.

The rest of this paper is organized as follows. In Section II we formulate and study the workload factoring problem using game theory. Next, in Section III we propose using job replication as a scalable method for job scheduling. In Section IV we propose a novel method for managing elasticity in a fully distributed fashion. Section V concludes this paper. The complete thesis can be found in [13]. The results of our research have been published in [10]–[12], [14], [16]–[18].

II. Workload Factoring with the Cloud – a Game Theoretic Perspective

Contention among users utilizing a single shared resource arises in multiple contexts of computing and computer communications. This contention may also occur when different users make use of a public cloud [5], [19]. Cloud users are granted access to a virtual machine, but eventually share the cloud provider's physical resources. As part of our research, we have conducted a set of experiments corroborating this claim [15].

We consider a setup where each service provider can use the cloud as well as a private resource in order to service some user request. Each service provider needs to decide how to split its job between the cloud and the private resource, such that the job's completion time would be minimized. Since a private resource can only be used by its owner, it provides guaranteed performance, i.e., it does not depend on the decisions nor job patterns of other service providers; the cloud, on the other hand, constitutes a more powerful infrastructure, yet its performance does depend on the loads presented by other service providers.

Since service providers can be expected to make decisions in a self-optimizing (selfish) manner, we face a non-cooperative game, whose investigation is the subject of this study.

We consider N players, each of which is a service provider that has a job that requires processing. Player strategies are the choice of the fraction of the job that will be processed by the cloud. We denote player i's strategy by σ_i. The strategy profile of all players is denoted by σ. The goal of each player is to minimize the expected completion time of its job.

Each player i is characterized by the size of its job, denoted by ω_i. Each player has its private resource, whose performance is manifested by a delay (e.g., computation time) function $L_i(\cdot)$. We assume that $L_i(\cdot)$ is strictly increasing in the amount of work (i.e., strictly decreasing in σ_i). In addition, we assume that $L_i(1) = 0$, i.e., if player i submits its entire job for processing by the cloud, the delay incurred by its private resource is 0.

We denote the estimated completion time of a job of player i by the shared resource by T_i^{shared}, noting that in general, it depends on the loads submitted to the cloud by the various players. We assume that T_i^{shared} is a sum of two components, namely $d_i(\sigma_i)$ and $D(\sigma)$.

The first component, $d_i(\sigma_i)$, is a delay component that is specific to the player and independent of the load submitted by other players to the shared resource, i.e., $d_i : \sigma_i \rightarrow \Re^+$. We assume that $d_i(\cdot)$ may be discontinuous at 0, but that it is continuous for every other value. Furthermore, we assume $d_i(\cdot)$ is monotonically non-decreasing and that $d_i(0) = 0$. For example, $d_i(\cdot)$ can correspond to the time required to

send a job to the cloud, which, in turn, may depend on the geographical location of the player with respect to the cloud's resources [21]. We note that transmitting nothing incurs no delay, but transmitting even a single bit may incur a non-negligible delay, hence our relaxation that $d_i(\cdot)$ may be discontinuous at 0.

The second component, $D(\sigma)$, is the cloud's processing delay, which depends on the particular loads submitted to it. Since the job sizes, $\{\omega_j\}_1^N$, are assumed to be constant, D is a function of the strategy profile σ, i.e., $D : \sigma \rightarrow \Re^+$. We assume that $D(\cdot)$ is continuous and strictly increasing (in every one of its parameters), and, in addition, $D(0) = 0$. We thus have:

$$T_i^{shared}(\sigma) = D(\sigma) + d_i(\sigma_i).$$

A player's completion time is the time in which its job completes processing (both by the private and the cloud), formally:

$$T_i(\sigma) = \max\left\{T_i^{shared}(\sigma), L_i(\sigma_i)\right\}.$$

Note that in case the player chooses not to send any work to the cloud, its completion time is $L_i(0)$.

We denote the shared resource's actual processing completion time by $T^{Shared}(\sigma)$. We note that $T^{Shared}(\sigma)$ may be different than the expected completion time perceived by the player, $T_i^{shared}(\sigma)$.

We note that for certain values of σ_{-i}, $T_i^{shared}(< \sigma_{-i}, \sigma_i >)$ is not a continuous function of σ_i since the completion time for $\sigma_i = 0$ (no computation by the cloud) may be much smaller than the completion time where a very small fraction is sent to the shared resource. This discontinuity prevents us from adopting "classic" results on the existence of Nash equilibria. Moreover, we note that each player may have a different $d_i(\cdot)$, which further complicates the analysis of Nash equilibria.

First, we prove that the WLF game admits a Nash equilibrium. As pointed out above, this task is especially challenging due to the discontinuity of the objective function as well as the heterogeneity among users in terms of the d_i component.

Definition 1: Let i be some player. Let σ denote a strategy in which $\sigma_i = 0$. We denote $L_i(0) \prec D(\sigma)$ if for every $\epsilon > 0$, $L_i(\epsilon) < D(< \sigma_{-i}, \epsilon >) + d_i(\epsilon)$.

Theorem 1: The 2-player WLF game admits a Nash equilibrium profile.

Proof: We show, by construction, that the game has a Nash equilibrium point.

We begin with a state in which neither of the players makes any use of the shared resource, i.e., $\sigma^0 =< \sigma_1^0, \sigma_2^0 >$, where $\sigma_1^0 = 0$ and $\sigma_2^0 = 0$. We proceed by having each player, in its turn, perform a best-response move.

Let σ_1^1 denote the relative amount of workload submitted by player 1 to the shared resource following the first best-response move. We consider two cases: if $L_1(0) \prec D(< 0, 0 >)$, player 1 would not make use of the shared resource in any case. Accordingly, a strategy profile in which $\sigma_1^1 = 0$ and player 2 is at best response constitutes a Nash equilibrium. We continue under the assumption that $L_1(0) \not\prec D(< 0, 0 >)$, and therefore $\sigma_1^1 > 0$.

Next, we consider player 2; if $L_2(0) \prec D(< \sigma_1^1, 0 >)$, it holds that $< \sigma_1^1, 0 >$ is a Nash equilibrium profile. Otherwise, it holds that player 2 will submit some non-zero relative workload, denoted by σ_2^1, for processing by the shared source. Consider, then, the latter case.

When player 1 performs a best-response move again, it is clear that $D(< \sigma_1^1, 0 >) < D(< \sigma_1^1, \sigma_2^1 >)$ (i.e., the workload submitted by player 2 to the shared resource increased the shared resource's delay). Thus, it holds that $L_1(\sigma_1^1) < D(< \sigma_1^1, \sigma_2^1 >) + d_1(\sigma_1^1)$, so player 1's best-response move would be to decrease its relative workload submitted to the shared resource (potentially, even to 0), i.e., $\sigma_1^2 < \sigma_1^1$.

In a similar fashion, when player 2 performs a best-response move again, it holds that $D(< \sigma_1^1, \sigma_2^1 >) > D(< \sigma_1^2, \sigma_2^1 >)$ (i.e., since player 1 reduced its relative workload in the shared resource, the shared resource's delay has decreased). Therefore, player 2's best-response move would be to increase it relative work load, i.e., $\sigma_2^2 > \sigma_2^1$.

Once again, since player 2 increased its relative workload processed by the shared resource, it holds that player 1 would decrease its as part of his next best-response move, i.e., $\sigma_1^3 < \sigma_1^2$; we can apply the same process repeatedly.

We conclude that the sequence of player 2's relative workloads following its best-response moves constitutes a strictly increasing sequence, $\{\sigma_2\}_1^k$. By definition, it holds that this sequence is upper bounded (by 1), and therefore converges to a non-zero value. We denote its limit by σ_2^*.

Similarly, it holds that the sequence of player 1's relative workloads following its best-response moves constitutes a strictly decreasing sequence, $\{\sigma_1\}_1^k$. By definition, it holds that this sequence is lower bounded (by 0), and therefore converges. We denote its limit by σ_1^*.

Finally, we conclude that $\sigma^* =< \sigma_1^*, \sigma_2^* >$ is a Nash equilibrium profile and the Theorem follows. ■

In [11], [17] we extend this proof to a case of any N players, and prove that the Nash equilibrium is unique.

We further explore the inherent inefficiency that stems from the players' noncooperative behavior. Specifically, we study the *price of anarchy* [8] of the game and indicate that it may heavily depend on the specific manner in which service providers perceive the performance they experience when using the cloud, as well as on the cloud's behavior under heavy load. For example, we demonstrate that, when players (service providers) make worst-case decisions (e.g., when handling time-critical jobs), the unique Nash equilibrium coincides with the social (i.e., system-wide) optimum; whereas in another case, where the cloud performs outsourcing under heavy load, employing techniques such as *cloud federation* [20], we demonstrate that the price of anarchy may be arbitrarily large.

We then turn to investigate practical implications of our formal model and analysis. An important question is whether proper design and management of the system can mitigate the deficiencies of noncooperative behavior. One deficiency, measured by the price of anarchy, is the sub-optimality of

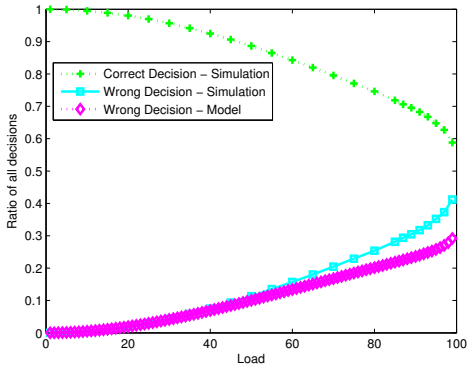

Fig. 1: Correctness of scheduling decisions as a function of the load when the load balancer chooses the processing server based on queue length

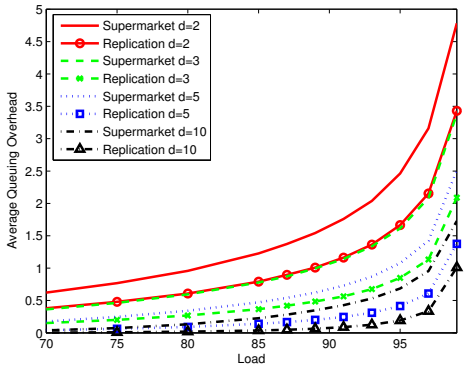

Fig. 2: Average Queuing Overhead, Poisson distributed job arrival times, exponentially distributed processing times, no delays, the Supermarket Model vs. replication, high load values

the social performance at the Nash equilibrium. Yet, from a practical standpoint, there are other deficient aspects that need to be considered, for example the potential "conquest" of the cloud by certain communities of service providers (e.g., "heavy users"), which may "scare off" all others.

Our study indicates that the employment of management tools can indeed come at rescue in some cases, yet, and somewhat counter-intuitively, in other cases they may fail to deliver. Specifically, we demonstrate the latter (negative) finding by showing that the employment of an apparently appealing admission control scheme deteriorates system-wide performance; whereas we demonstrate the former (positive) finding by showing how the partition of the cloud into "virtual sub-clouds" can result in increasing its attractiveness to various types of service providers.

III. Replication-based Load Balancing

Load balancing of large distributed server systems is a complex optimization problem of critical importance in cloud systems and data centers. Existing schedulers often incur a high communication overhead when collecting the data required to make scheduling decisions, hence delaying job requests on their way to the executing servers.

To avoid the "stale data problem" [2], the data must be collected as close as possible to the arrival of the job. Thus, common load balancing approaches, such as the "Supermarket Model" [9], invoke the data collection process upon a job's arrival. The Supermarket Model is a scheduling policy where, upon the arrival of a new job, d randomly selected servers are queried for their queue length, and the job is sent to the server with the least number of jobs. Collecting data to support the scheduling decision introduces a delay in the execution of the job, as the data collection process hinders the selection of the executing server and the arrival of the job to that server.

The size of data centers, coupled with their distribution across the globe, call for fully distributed load balancing techniques. Accordingly, it is important to understand how well can an oblivious[2] distributed load sharing system perform.

As part of our research [12], [16], [18], we propose a novel load balancing scheme that minimizes the time a job spends till being assigned to a server and, in addition, scales well. Our approach is based on duplicating each job request into several replicas, which are sent to randomly chosen servers. We assume that servers process jobs according to a first-come-first-served policy. Servers assigned with replicas of the same job coordinate the removal of all copies but one. Specifically, when a replica arrives to the head of the queue and begins processing, a signal is sent to the other corresponding servers in order to trigger the removal of all other replicas.

Unlike the Supermarket Model, which chooses the processing server based on the shortest queue policy, our scheme relies on actual completion times, thus reducing the queuing overhead in the server. We analyze the cases where the Supermarket Model errs when selecting the shorter queue. Our results, based on our analysis as well as on simulations and depicted in Figure 1, indicate that, when basing the scheduling policy on queue lengths, the ratio of erroneous decisions rises with the load to a considerable level. For example, for loads above 90%, some 30% of the decisions are wrong. That is, when the system particularly needs the load balancing mechanism to perform at its best, a queue-lengths-based mechanism performs at its worst.

We test our technique under three load profiles. The first is a "traditional" setting in which jobs arrive based on a Poisson process, and job lengths (required processing times) are exponentially distributed. Such load profiles have been widely considered in the literature, due to their amenability to formal analysis as well as often being suitable approximations of reality.

We further test our scheme under a load profile in which jobs arrive based on a Poisson process but their lengths abide to a bounded-Pareto-distributed process [4], with very high variance. In such a setting, in addition to incurring delays in the scheduler, some jobs may be unfortunate to be scheduled for execution on a server behind a very long job.

Finally, we test our techniques based on HTTP traces collected from real systems. For this purpose, we follow on the work presented in [3] and develop a workload that mimics the behavior of a social network site, such as Facebook.

Our results, a sample of which are presented in Figure 2 and Figure 3, show that the replication scheme improves the selection of the processing server, this offering a significant improvement in queuing overhead when compared to the

[2] An *oblivious system* (also termed *static system*) is a system that is independent of the current state, and does not use any dynamic input.

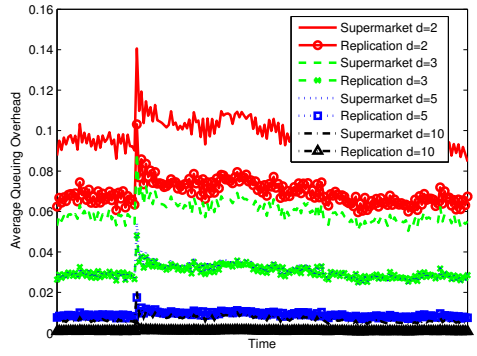

Fig. 3: Average queuing overhead, job arrival times are based on a trace of over eight million HTTP request to Wikipedia [22], no delays, the Supermarket Model vs. replication

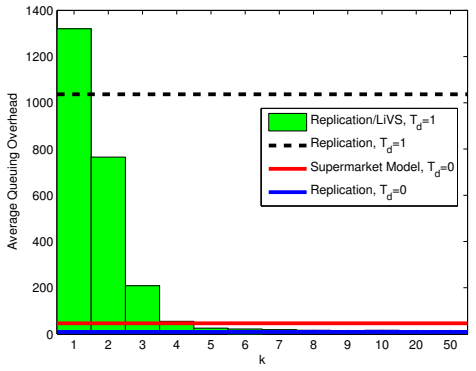

Fig. 4: Average queuing overhead with Location-in-Vector Selection, Poisson distributed job arrival times, Bounded-pareto distributed processing times, the exponent of the power-law, $\alpha = 1.4$, maximal job length, $H = 10,000$, $d = 2$, $load = 97\%$, signal propagation delay, $T_d = 1$

Supermarket Model.

Yet, the time required for the removal signals to arrive from the sending server (the one executing the job) to the servers holding the other replicas may hinder the performance gain offered by our scheme. Indeed, if replicas arrive at the heads of the servers' queues approximately at the same time, then the servers may spend time processing jobs that are doomed to be removed. Thus, in the presence of very large delays (with respect to the job's processing time), multiple copies of the same job might complete processing at different servers.

Surprisingly, our results indicate that in the case of bounded-Pareto distributed job lengths and a low α value, signal propagation delay has little to no effect on system performance. However, in other cases, the usage of a high number of replicas, or a moderate signal propagation delay, may severely degrade the performance of our scheme.

Accordingly, we developed a heuristic solution that improve our scheme in the presence of signal propagation delay. Our heuristic, called *Location-in-Vector Selection* (LiVS), modifies the job selection policy at the server. Specifically, instead of processing jobs based on the first-come-first-served policy, servers pick the next job out of the first k jobs in the queue (where k is a parameter of the heuristic) based a simple prioritization policy. We evaluate the LiVS heuristic through simulations and show that it can be used to restore the high performance gain offered by our scheme under moderate signal propagation delay. One exemplary result demonstrating the effectiveness of the LiVS heuristic is depicted in Figure 4.

IV. A Distributed Approach for Cloud Elasticity

One of the important concepts behind the adoption of cloud computing is the Pay-As-You-Go model. In this model, which is currently in use by major cloud providers such as Amazon [1] and Microsoft [23], service providers pay only for allocated resources and the amount of these resources can be dynamically modified. For example, paying per VM (according to the specification of the VM) is done only for the duration of the VM's lifetime.

However, this model places a major dilemma to the service providers, namely - how much resources to acquire? Indeed, on the one hand, a higher amount of resources leased from the cloud results in better service quality; but on the other hand, a higher amount of resources incurs higher operational expenses, as the service provider has to pay the cloud owner for the amount of requested resources. In other words, while increasing the amount of resources used by the service has the potential of increasing its income, over-provisioning may lead to decrease in revenue.

One of the common mechanisms used to address this challenge is *elasticity*, i.e., the ability to dynamically adjust the amount of the resources allocated to the service based on the demand for that service. This capability is used, for example, by on-line shopping service providers in order to expand their service around the end of the year when demand rises as people go on-line to do their holiday shopping; when the holiday season is over and demand drops, the service providers can scale down their service and release the resources back to the cloud. Another example, where resource allocation should be adjusted in a much shorter time frame, is a case where a large-scale disaster occurs and users log in to report their experience or check on their relatives and friends. In such a scenario, the demand for social network services may increase rapidly and unexpectedly during a short period of time, thus the amount of resources allocated to the service should be adjusted accordingly in order to maintain the desired user experience.

Typically, the dynamic adaptation of the allocated resources is accomplished by monitoring their state. However, for large cloud-based services, tracking the performance of each server or VM and monitoring each user request is impractical. In such cases, no centralized entity has full knowledge regarding the state of the system and current techniques no longer work.

We propose a novel resource management scheme that enables the service provider to lease the "right amount" of computational resources (e.g., the right number of VMs), in view of the above fundamental tradeoff and in the presence of dynamic workload together with a highly distributed environment. Our scheme comprises of two components, namely: a task assignment policy and a VM management policy. We emphasize that each of these components should work in a distributed manner without assuming full knowledge of the overall system state. This creates an interesting coupling between these components, since poor load balancing may result in an overloaded component that may issue a request for (globally unneeded) additional resources.

Our task assignment policy strives to "pack" VMs with as many jobs as possible, while ensuring that they remain within

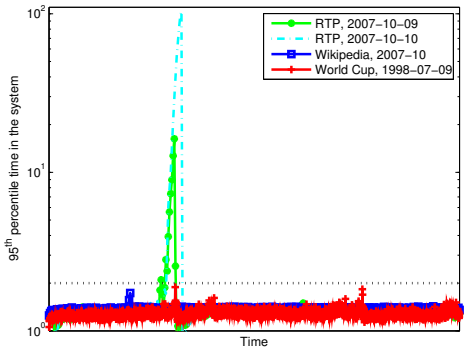

Fig. 5: 95^{th} percentile time in the system for HTTP traces obtained from [6], [22]. Packing threshold, $T^P = 5$, activation threshold, $T^A = 8$, VM instantiation time=120 seconds

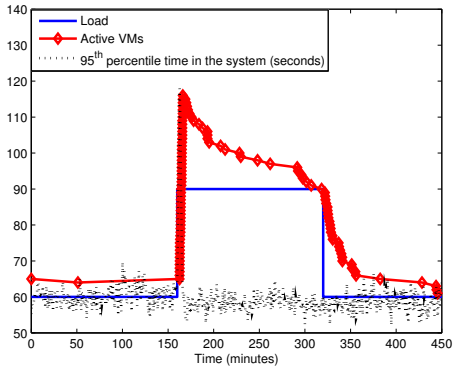

Fig. 6: EC2 test, sharply changing load. Packing threshold, $T^P = 5$, activation threshold, $T^A = 8$, VM instantiation time=120 seconds, idle time before VM termination=160 seconds

operational bounds, i.e., that all the requests can be serviced within the SLA requirements.[3]

Fundamentally, our scheme follows the classic Supermarket Model [9]; upon the arrival of a new job to the system, some d VMs are sampled uniformly at random for their load (measured through the number of jobs handled by the sampled VM). However, instead of sending the job to the least loaded VM (as the Supermarket Model would do), we send it to the *most loaded* among the sampled VMs that are within the operational bounds. Only if all sampled VMs are loaded beyond the operational bounds, we send the job to the least loaded one. Similarly to the Supermarket model, our task assignment policy admits a fully distributed implementation [2].

Our VM management policy calls for the instantiation of a new VM whenever all sampled VMs are over-loaded. Furthermore, we adopt the distributed policy defined in [3] (in the realm of power management) to determine when to release VM resources back to the cloud, namely: if a VM completes the processing of all jobs assigned to it, and receives no new jobs for some time, then it turns itself down, thus releasing all resources back to the cloud provider. Due to the "packing" property of the task assignment policy, each VM can make this decision by itself, thus not requiring any centralized control.

To fully investigate the applicability of our scheme we test it with a variety of load patterns, including four patterns based on real HTTP traces, and three synthetic loads exhibiting a fixed, a gradually changing and a sharply changing arrival rate. Furthermore, in addition to simulation, we implemented and tested our scheme within Amazon's EC2 framework [1] emulating a real-life commercial environment.

We show that, in such a completely distributed setup, our scheme is able to adapt to changing load, by adjusting the number of active VMs as needed in order to meet the SLA goals under the new load. When the load rises, numerous arriving jobs sample over-loaded VMs, and new VMs are instantiated. For example, Figure 5 depicts the 95^{th} percentile of response time for a variety of HTTP traces obtained from real world websites [6], [22]. The figure shows that the 95^{th} percentile of response time is well below the target threshold (marked by a black dashed line in the figure), excluding a few points where the load sharply changes (the HTTP traces are described in more detail in [13]).

We show that, even under a very drastic change in load, our scheme quickly adapts to the new state and the time period in which SLA requirements are violated is very short. We note that such a time period is on account of the time required to instantiate a new VM, and cannot be avoided without significantly over-provisioning, and consequently increasing the operational costs of the service. Figure 6 depicts the results of an EC2-based test running a sharply changing load. Results show how quickly the system adapts to the load, meeting the SLA target (of 80 seconds) while keep the number of active VMs to a minimum.

V. Conclusions

Overall, our results show that taking into consideration key aspects of large scale systems, such as the users inability to coordinate, as early as in the design phase of the system may have detrimental impact of the cloud's performance. On the other hand, a robust system design, coupled with some of the suggested resource management techniques, may support scaling the system with minimal to no performance degradation.

Another part of this thesis, addressing problems in the realm of the design and control of networks, has been published in [10], [14].

[3]We assume that the service provider needs to comply with a Service Level Agreement (SLA) that specifies a target maximum time that a job spends in the system and the allowable rate of violations from this target.

REFERENCES

[1] Amazon Elastic Compute Cloud, http://aws.amazon.com/ec2, visited: 2014-07-07.

[2] D. Breitgand, R. Cohen, A. Nahir, and D. Raz, "On cost-aware monitoring for self-adaptive load sharing," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 1, pp. 70–83, 2010.

[3] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems*, vol. 30, no. 4, p. 14, 2012.

[4] M. Harchol-Balter, "Task assignment with unknown duration," *J. ACM*, vol. 49, no. 2, pp. 260–288, 2002.

[5] B. Hayes, "Cloud computing," *Commun. ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[6] IRCahce, http://www.ircache.net/, visited: 2013-10-26.

[7] P. D. M. Jr., F. de Figueiredo, D. Maia, F. V. Brasileiro, and A. Coelho, "On the planning of a hybrid IT infrastructure," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, 2008, pp. 496–503.

[8] E. Koutsoupias and C. H. Papadimitriou, "Worst-case equilibria," *Computer Science Review*, vol. 3, no. 2, pp. 65–69, 2009.

[9] M. Mitzenmacher, "How useful is old information?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 1, pp. 6–20, 2000.

[10] A. Nahir, A. Orda, and A. Freund, "Topology design of communication networks: A game-theoretic perspective," *Networking, IEEE/ACM Transactions on*, vol. 22, no. 2, pp. 405–414, 2014.

[11] A. Nahir, A. Orda, and D. Raz, "Workload factoring: A game-theoretic perspective," *Networking, IEEE/ACM Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[12] ——, "Replication-based load balancing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, 2015.

[13] A. Nahir, "Design and management of complex distributed systems: Optimization and game-theoretic perspectives," Ph.D. dissertation, Technion, Israel Institute of Technology, 2014. [Online]. Available: http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2014/PHD/PHD-2014-07

[14] A. Nahir, A. Orda, and A. Freund, "Topology design and control: A game-theoretic perspective," in *INFOCOM '09. Twenty Eighth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2009, pp. 1620–1628.

[15] A. Nahir, A. Orda, and D. Raz, "Workload factoring: A game-theoretic perspective," Department of Electrical Engineering, Technion, Haifa, Israel, Tech. Rep., 2011. [Online]. Available: http://www.ee.technion.ac.il/Sites/People/ArielOrda/Info/Other/NOR11CWF.pdf

[16] ——, "Distributed oblivious load balancing using prioritized job replication," in *Proceedings of the 8th International Conference on Network and Service Management*, ser. CNSM '12, 2012, pp. 55–63.

[17] ——, "Workload factoring with the cloud: A game-theoretic perspective," in *INFOCOM '12. Thirty First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2012, pp. 2566–2570.

[18] ——, "Schedule first, manage later: Network-aware load balancing," in *INFOCOM '13. Thirty Second Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2013, pp. 510–514.

[19] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *ACM Conference on Computer and Communications Security*, 2009, pp. 199–212.

[20] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. S. Montero, Y. Wolfsthal, E. Elmroth, J. A. Cáceres, M. Ben-Yehuda, W. Emmerich, and F. Galán, "The reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, p. 4, 2009.

[21] A. Wang, C. Huang, J. Li, and K. W. Ross, "Estimating the performance of hypothetical cloud service deployments: A measurement-based approach," in *INFOCOM '11. Thirtieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2011, pp. 2372–2380.

[22] WikiBench - Wikipedia Access Traces, http://www.wikibench.eu/?page_id=60, visited: 2013-10-26.

[23] WindowsAzure - Microsoft's Cloud Services Platform, http://www.microsoft.com/windowsazure/, visited: 2014-07-07.

[24] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Intelligent workload factoring for a hybrid cloud computing model," in *Proceedings of the 2009 Congress on Services - I*, ser. SERVICES '09, 2009, pp. 701–708.