Examine the Advantages of An Integrated Scalability Approach at Various Cloud Stack Layers

SeyedEbrahim Dashti and Ahmed Rahman Abdulzahra
Sayed.dashty@gmail.com

Department of Computer Engineering, Jahrom Branch, Islamic

Azad University

Abstract
The development of cloud computing has significantly altered how services are built, deployed, and made accessible to users outside of the organization. In actuality, the pay-as-you-go model of dispersed IT supported by the cloud computing paradigm calls for the outsourcing of software services and applications. In this situation, the capacity to ensure effective cloud performance management and to facilitate automated scalability become fundamental prerequisites. Users of the cloud are becoming more and more interested in a transparent and coherent image of the cloud, where performance is guaranteed in a variety of situations and under a variety of loads. In this essay, We examine the advantages of an integrated scalability approach at various cloud stack layers, concentrating on the database and compute infrastructure layers. In order to achieve this, we offer various performance measurements and a set of rules based on them to assess thecloud stack's condition and scale it as needed to maintain stable performance. Then, using a proof-of-concept architecture, we empirically investigate three scaling scenarios for cloud performance: database only, computing infrastructure solely, and the scenario where computing infrastructure and database compete for resources.
Keywords:
scalability, cloud, stack.

1. Introduction

The preferred method for providing IT services is moving toward the cloud [1]. The cloud paradigmoffers its users (i.e., end users and service providers) a number of benefits, including the ability tooutsource a portion of their operations to the cloud (for which strong IT skills are required), a decrease in the cost of owning, operating, and maintaining computational infrastructures, an increase in flexibility, and access to a scalable infrastructure. The spread of cloud technologies and solutions, on the other hand, leads to the deployment of numerous heterogeneous multi-layer cloud stacks [2]. New methods for performance monitoring and automatic scaling that are independent of cloud configuration and operate at many layersare becoming more and more in demand in this context.

In order to analyze the performance of several technologies offering functionality for a given cloud layer, existing performance evaluation and automatic scalability methodologies typically focus on a single cloud layer at a time [3], [4]. As an illustration, various NoSQL databases are contrasted to assess their functionality and support for scalability in various scenarios [3], [5], [6]; The same holds true when comparing various cloud infrastructures (IaaS) [4]. In this way, even when multiple layers of the cloud stack compete for the same physical resources, inter-layer synergy and interference effects are not takeninto account. Additionally, many current solutions presuppose an endless supply of resources that scale up on demand without any

Manuscript received March 5, 2025
Manuscript revised March 20, 2025
https://doi.org/10.22937/IJCSNS.2025.25.3.6

constraints (e.g., [4], [7]). But as mentioned in [8], This is not always the case, particularly in situations where there are limited IT resources (as in a private cloud) or financial resources (as in a hybrid/public cloud). This situation prevents us from treating each layer of the cloud stack as an independent building block; instead, it forces us to prioritize which layer to grow first when resources are scarce and many levels (such the database and computing infrastructure layers) need to scale.

In this article, we address the aforementioned issue and propose an integrated approach to cloud scalability that centers on a scenario in which the database and computational infrastructure layers are in competition for resources. Our contention is that blind scaling is inapplicable to the cloud settings of today, necessitating the definition of scalability solutions that take into account the possibility that the demands of the entire cloud stack cannot be met by the resources at hand. First, our method defines a set of indicators that can be assessed and quantified using commercial and open source software, as well as a set of guidelines for automatic scaling and performance monitoring that apply to both the database and computinginfrastructure levels. Then, it is applied to three scalability scenarios: database only, computing infrastructure solely, and a mix of the two where database and computing infrastructure fight for resources. A private cloud architecture is used to experimentally examine the three scenarios. The rest of this essay is structured as follows. Section II provides examples of our reference architecture and motivating scenario. The metrics and guidelines for automatic scalability are presented in Section III. Section IV details the results of our experiment. The relevant study is summarized in Section V, and our conclusions are presented in Section VI.

II- MOTIVATING SCENARIO AND REFERENCE ARCHITECTURE

Our inspiring scenario and reference structure are presented in this section.

A) Motivating Scenario

Leading European telecommunications provider Telecom Italia offers its customers a range of mobile and cloud services. The Telecom Italia cloud computing offering known as Nuvola Italiana, or the "Italian Cloud," consists of a variety of services that assist businesses in the distribution of their applications and the implementation of their business processes. One of the most demanding needs in this situation is the ability to offer an autonomous scalability solution that targets the cloud stack at various stages. . Since applications typically need resources for service execution and delivery as well as resources for database operations (such as the conventional NoSQL map- reduce operations), many of the provisioned services, in particular, require scalability at least of the computing infrastructure and database [9].

Here, we focus on data-intensive applications (such as data-intensive websites developed in Ruby and deployed on the cloud1), which demand on two distinct pools of virtual machines because business logic and data layers are separated (VMs). First, we specifically concentrate on conventional cases where the business logic (i.e., computational infrastructure) and the data (i.e., database) layers are scaled independently depending on the loads of requests. Then, we concentrate on a scenario where the two levels compete with one another for resources. The research presented in this paper tries to address the following scalability requirements.

Elasticity scaling. The pay-as-you-go paradigm must be taken into consideration when discussing cloud scalability, and elastic approaches that scale up, out, and down based on the real load must also be supported.

Reactivity vs Proactivity. Scalability must accommodate both proactive and reactive strategies. By using appropriate metrics (such as CPU and memory use), a reactive method to scalability assesses the status and configuration of the cloud and scales as necessary in accordance with rules concerning those metrics. In order to decrease the likelihood of situations when the stack is overloaded or underloaded and a reactive approach is required, a proactive method monitors the status and configuration of the cloud and preemptively scales.

Multi-layer. Clouds are inherently multi-layered and diverse. A cloud scalability strategy must address the heterogeneity of the cloud and support various methods that enable scaling at various layers of the cloud architecture.

Resource limitations. A scalable infrastructure with boundless resources that are accessible on demand is frequently how the cloud is viewed. This isn't always the case, though [8], and as a result, a scalability strategy that optimally assigns/revokes resources when and when needed is required.

B) Typical Architecture

The layers in our reference architecture's cloud stack are as follows: the management of infrastructure resources (such as compute, network, and storage) through the deployment of components providing IaaS functionalities; the management of application lifecycle and scaling rules through the deployment of components providing PaaS functionalities; and the implementation and management of databases through the provision of database infrastructure. Our approach is based on Telecom Italia's cloud offering, which covers the following group of products.

- OpenStack. IaaS open source software that enables management and monitoring of infrastructure resources It establishes infrastructure templates for VM provisioning, controls network and storage operations, and provides a number of APIs that expose monitored data to the architecture's higher layers.

- Cloudify. a PaaS system that is open source and allows management of the application lifetime. In order to offer scalability at the infrastructure layer, it implements a collection of recipes that are mapped on OpenStack templates. It offers a list of supported application architectures, supports metrics definition, and scaling rules (i.e., Apache, Tomcat, and MongoDB in our scenario).

- MongoDB. a document-oriented, NoSQL database with no schema. It is made up of three basic parts: a router service that receives and distributes requests to shards; shard servers that store data; configuration servers that store database configurations.

- KVM. It is the hypervisor that OpenStack uses to operate within the computing node.

We observe that OpenStack has been set up in a multi-node configuration, with three virtual machines serving as the controller, network, and computing nodes, respectively. We should also mention thatVMWare ESXi 5.5 is the hypervisor used to deploy the entire system.

III- PERFORMANCE METRICS AND SCALABILITY RULES

A strategy for automated scaling must first define a set of metrics tracking the condition of various cloud stack levels (Section III-A). Following that, a set of rules that can support automatic scaling at various layerscan be implemented using these metrics (Section III-B). Database layers and computer infrastructure are the main topics of this section.

A. Performance indicators

Performance metrics are numerical indicators of a certain cloud stack's health and can be tracked to inform various scalability strategies according to the scenario under consideration. We make a distinction between reactive and aggressive scaling. First, metrics for reactive scalability provide a quick snapshot of the current state of a particular computing infrastructure and/or database, enabling prompt responses to situations where an event may have an impact on the performance and availability of a certain configuration. We take into account the subsequent indicators for reactive scalability:

• CPU Load (CL): It gauges the actual CPU usage of a group of virtual machines that are assigned to a system. For instance, high levels of CL can indicate that a system is having trouble addressing a certain setof requests.

Memory Occupancy (MO) is a measurement of how much memory a pool of virtual machines given to a system actually uses. For instance, high values of MO can indicate that a system is having trouble handling situations when applications need to manage large amounts of data.

• Network Utilization (NU): This metric assesses how well network bandwidth is being used. High NU values, for instance, may indicate situations where the distribution of data in a database is not ideal, necessitating numerous data exchanges (such as for map-reduce processes), or they may indicate the need to reallocate resources from outside the system to handle a spike in demand.

• Host Availability (HA): This measure determines how many VMs are available and reachable via the network.

• Response Time (RT): the period of time between the dispatch of a request and the client's receipt of the appropriate response.

• Execution Time (ET): This metric gauges how long it takes for a service to respond to a request after receiving it.

• Request Ratio (RR): This metric compares the proportion of requests that are directed at various cloud stack layers.

Metrics for proactive scalability must also be established. These metrics analyze the performance trend and system evolution over a predetermined time window with a size of t seconds in order to predict and deduce potential demands for extra resources. Preemptive scaling minimizes potential service performance degradations and enables long-term maintenance of a constant performance level. If recurring patterns of computing infrastructure/database usage are present, these metrics are more useful. For proactive scalability, we take the following metrics into account:

• Request Rate (ReqR): This metric counts how many requests have come in within the specified time period. It can be used to recognize a specific pattern and subsequently estimate future resource scalability needs.

• Request Faults (RF): This metric evaluates the proportion of requests that were not fulfilled within a given time frame, allowing for the detection of systemic issues.

• Request Type (ReqT): It calculates the frequency of each type of request over a specified time period. Asan illustration, we can keep track of the rate of MongoDB queries per type (such as read, insert, and update) and respond appropriately. The measure Value can also be affected by the complexity of incoming requests, which can be assessed using the service description [10] or professional recommendations.

• CPU Load Pattern (CLT): It tracks the trend of CPU usage throughout the selected time period. It can be used to monitor the system's evolution and respond appropriately.

Similar to CLT, but for memory occupancy, is the Memory Occupancy Trend (MOT).

• Network Utilization Trend (NUT): similar to CLT but for network usage

The state of the database layer and the computing infrastructure can both be assessed using the aforementioned metrics. Additionally, they can be combined to provide more in-depth information, such as the current CPU Load with a rising or falling trend.

B) Scalability rules

To track the performance of cloud-based systems and choose the best strategy for automated scaling, we created an architecture based on the metrics in Section III-A. When specific events are detected that may have an impact on the system's performance and availability, our monitoring infrastructure will set off one ormore scaling rules. These rules simulate situations in which a single metric (or a combination of them) surpasses predetermined thresholds and causes the execution of specified scalability actions. To extract data from the underlying system and calculate performance metrics, our monitoring infrastructure I builds on specific APIs, in this case those provided by OpenStack and MongoDB through JMX, and ii) is based on Cloudify. Scalability rules are implemented as Cloudify recipes, which manage the scale out and scale down of the application.

We should point out that scaling up is not an option here because it is less successful than a scale out strategy [8].

Table I shows a set of rules that, when certain high/low thresholds are crossed, initiate scaling actions at the infrastructure and database layers by monitoring and evaluating performance metrics. Expert users can define thresholds based on prior tests and the domain under consideration. Table I lists the metrics to be combined for each rule (Formula) and describes the scalability steps (Action). We observe that a single metric's value is normalized from 0 to 1 using either the highest value it is capable of reaching or the highest value already recorded.

Details are provided in Table I(a), where metrics are used to assess the state of resources that are either assigned to the database layer (poolI) or the computing infrastructure layer (poolI) (poolDB). When the system encounters a major change in infrastructure performance, or, to put it another way, when pertinent formulas assume values above/below the high/low thresholds, an infrastructure-level rule is activated.Similar to this, when pertinent formulas indicate the requirement for scaling database resources, a database- level rule is activated for the database layer. Table I(b), instead, presents composite rules that monitor metrics referred to both poolI and poolDB. The table's scalability rules can then be applied as Cloudify recipes (see for example the excerpt in Figure 1).

There is a complexity to take into account when many recipes apply, request more resources, and must be processed simultaneously. These recipes may insist on various cloud layers. As was mentioned in Section II, there may be situations in which all triggered rules are competing for the same set of resources and cannot be implemented simultaneously owing toresource and/or financial limits. In this case, we suggest augmenting our scalability solution with a priority-based strategy that gives each rule a priority. If further rules demanding ascale out activity are present, they are applied in accordance with the priority until no more resources are available or all rules have been applied. It is significant to highlight that Cloudify's monitoring infrastructure must be expanded to handle this situation because the existing implementation does not provide the selective execution of distinct recipes based on priority. We experimentally evaluated a situation in Section IV-D where Rule 1 and Rule 2 are both triggered and require an additional VM to be assigned to poolI and poolDB, butthere is only one VM that is accessible. This analysis was done to demonstrate the possibilities of our technique. Our tests demonstrate which option provides the greatest performance advantages to establish Rule 1's precedence over Rule 2.

Finally, Figure 1 depicts a Cloudify recipe that implements Rule 1 from Table I. CLI is first assessed using script statistics. a CLInfr that calculates the value of relevant data after retrieval. If the ratio surpasses the high threshold, the system then adds one VM outfittedwith a Tomcat instance to poolI (instancesIncrease) (0.75). Conversely, if the ratio fallsbelow the low threshold, it removes one instance (decreaseInstance) (0.25). Every 20 seconds, CLI is assessed (movingTimeRangeInSeconds

Table I SCALABILITY RULES

Rule	Formula	Action
1	CL_I	Status of $pool_I$ in terms of CL_I. High (Low) values suggest to increase (decrease) $pool_I$.
2	CL_{DB}	Status of $pool_{DB}$ in terms of CL_{DB}. High (Low) values suggest to increase (decrease) $pool_{DB}$
3	CL_I *ET	Status of $pool_I$ in terms of CL and ET. High values indicate an increasing ofexecution times due to critical infrastructure operations, suggesting to increase $pool_I$.
4	CL_{DB}*ET	Status of $pool_{DB}$ in terms of CL and ET. High values indicate an increasing ofexecution times due to critical database computation, suggesting to increase $pool_{DB}$.
5	$HA_I/[HA_I+(CL_I*ReqR)]$	Status of $pool_I$ in terms of no. of available VMs w.r.t. the status of the system and the actual request trend. Low values indicate that the system needs an increase in resources of $pool_I$.
6	$HA_{DB}/[HA_{DB}+(CL_{DB}*ReqR)]$	Status of $pool_{DB}$ in terms of no. of available VMs w.r.t. the status of the system andthe actual request trend. Low values indicate that the system needs an increase in resources of $pool_{DB}$.

IV- EXPERIMENTAL EVALUATION

To test the viability of our strategy, we created the following proof-of-concept private cloud architecture, focusing solely on reactive scalability. First, we studied a case in which the database (Section IV-C) or computational infrastructure (Section IV-B) requires greater resources. Then, we looked at a variety of the prior scenarios in which the database and computational infrastructure both had to scale up and compete for scarce resources (Section

IV-D). We made the assumption that data-intensive apps would use separate pools of virtual machines for infrastructure and databases.

A) Experimental setting

Two physical servers were used in a cluster for testing. The first server (Server 1) is a Dell Precision T1650 with an Intel Xeon Quad Core 2.6 GHz processor, 32 GB of RAM, a 1 TB hard drive spinning at 7200 RPM,and two 1 Gb/s Ethernet NICs. The second server (Server 2) is an Acer Veriton M6620G with an Intel Core- i7 3770 3.40 GHz processor, 16 GB of RAM, a 1 TB hard drive spinning at 7200 R On two different poolsof virtual machines, one for the database and the other for the computing infrastructure, the components of our reference architecture (see Section II-B) have been installed and set up as follows.

Regarding the computing infrastructure, OpenStack has been set up and tested using three virtual machines (VMs): two of the machines house the control and network nodes and are deployed on Server 2, while the third computer houses the compute node and is installed on Server 1. The compute node controls a collection of virtual machines (VMs), each of which has a Tomcat instance and 1 vCPU and 1GB of RAM. Regarding the database, we considered MongoDB (in a multi-shard configuration) made up of three component types (i.e., shard servers, configuration servers, and router servers), each of which was installed on a virtual machine (VM) located on Server 1 and furnished with a single virtual CPU and one gigabyte of RAM.

These hybrid MongoDB configurations are typical of Ruby-based data-intensive services and websites. We installed Cloudify and connected its shell to OpenStack for IaaS administration to finish our proof-of- concept cloud stack. Then, using Cloudify, we made a virtual machine (VM) for managing OpenStack and one for every Tomcat server and MongoDB shard.

Then, we recreated a real-world scenario with multiple clients making requests simultaneously (i.e., threads).Using Apache JMeter, an open source functional testing solution for services, the demand on the computing infrastructure has been generated and supplied to two different web apps, deployed on the Tomcat server.The first program is a straightforward hello world (hw). Following the Gregory-Leibniz series, the second application (hw) expands hw to calculate the value with a precision of 70,000 digits; hw allows us to primarily stress the CPU of our physical infrastructure. Using the Yahoo! Cloud Serving Benchmark(YCSB)

(https://github.com/brianfrankcooper/YCSB/wiki/), a framework and a set of workloads frequently used for evaluating and benchmarking the performance of various NoSQL databases, the load on thedatabase has been generated and sent to the MongoDB cluster connected to our applications. In each experiment, the average Transactions Per Second were measured (TPS).

B) Computing infrastructure scaling

We evaluated our strategy under various load fluctuations to determine whether performance would increase as the computing infrastructure scaled out (i.e., a single VM is added to poolI). We used the recipe implementing Rule 1 (Figure 1) to manage infrastructure scalability when either hw or hw is taken into account in order to keep things simple without sacrificing generality. Each test case: I took into account 20 active clients (threads) sending requests; ii) simulated different loads, varying the request per second (rps) in 500rps, 1000rps, 1500rps, 2000rps, 2500rps, 3000rps, 3500rps, 4000rps (3500rps and 4000rps for hw only), and iii) produced a total number of 100,000 requests.

Figure 2 illustrates the changes in TPS depending on whether hw or hw is taken into account. In both cases, we started with a fundamental architecture—a single system running Tomcat—and scaled it up to four virtual machines. Figure 2(a) demonstrates that when 4 VMs are deployed with application hardware, the system can handle a maximum load of 1896 TPS for 4000rps. With compared to configurations with 1 VM,2 VMs, and 3 VMs, respectively, we saw improvements of 39.5%, 28.7%, and 7.1% in this case. However, aconfiguration with 1 or 2 VMs yields the best performance for situations with 1500, 2000, and 2500 RPS.

. This is mostly caused by the complexity added by managing several VMs and by the properties ofhardware that responds after very little computation. Figure 2(b) demonstrates that when 4 VMs are deployed with application hardware, the system can handle a maximum load of 1650 TPS for 2000rps. With compared to configurations with 1 VM, 2 VMs, and 3 VMs, respectively, we were able to boost performancein this case by 53.6%, 27.2%, and 12.2%, respectively, following a similar pattern to the one seen for the hardware.

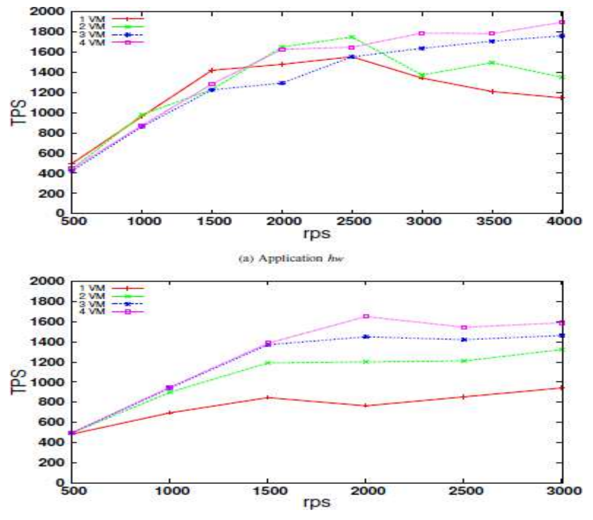

Figure 2. Performance evaluation (TPS)

A) Database scaling

We examined our strategy under various loads to evaluate performance gains as the database layer scalesout. We used a single recipe (implementing Rule 2 in Table I), which adds a VM to poolDB when a decline in performance is noticed, same to infrastructure scaling. Each test case generated 100,000 requests while simulating various loads with rps of 50, 100, 200, and 400. Three scenarios with different database request types were examined for each load: 5% read, 95% update; 95% read, 50% update; and 50% read, 50%update (RU0595). In each case, we expanded up to 4 shards from the initial basic MongoDB configuration with 1 shard (i.e., 1 VM) (i.e., 4 VMs) [26] .

Figure 3 depicts the changes in TPS that occur when poolDB scales out in situations I through iii). First, we should point out that poolDB can support more TPS the more VMs are provided for it. The database wasable to handle a load of 50 rps in all cases, we then note. Additionally, in scenario I as shown in Figure 3(a), the system could handle a load of no more than 90 TPS when configured with just two VMs, and this number increased to 131 TPS when four VMs were used. In case ii), as depicted in Figure 3, (b), In the configuration with two VMs, the system could handle a maximum load of 212 TPS; this number rose to 399 TPS with four VMs. In this instance, when 400 rps are put into the database, MongoDB is able to handle all the queries thanks to 4 VMs. The outcomes of cases I and ii) demonstrate MongoDB's capacity to significantly boost performance when read operations outnumber update operations. In scenario iii), where we pressured the DB infrastructure sending primarily update queries, this point is made more obvious. MongoDB could handle a maximum of 79 TPS with just 2 VMs and 121 TPS with 4 VMs, as shown in Figure 3(c). In conclusion, we observed increases of 46% in scenario I, 88% in scenario II, and 53% in scenario III when we increased our poolDB from 2 VMs to 4 VMs [26]

B) Competitive Scaling

We took into account a situation where poolI and poolDB needed to grow out simultaneously but had a finiteamount of resources. To do this, the following three web applications were deployed: I) the hardware used inearlier experiments; II) an extension of the hardware called hw,r that makes it possible to send read requests to MongoDB for each request received; and III) an extension of the hardware called hw,u that makes it possible to send update requests to MongoDB for each request received.

Then, under various rates of rps generated as follows, we evaluated the performance benefits when the extra VM is either assigned to poolI or poolDB. A rate rd=•ri of rps is generated and sent to hw,r (95%) and hw,u (5%), given a rate ri of rps delivered to hw. In particular, we first altered rd in 100rps and 400rps whilefixing ri to 1000rps. Then, we varied ri between 1500, 2000, and 2500 rps while fixing rd to 200 rps.

Figure 4 illustrates the rise in TPS that results from adding a single VM to poolI or poolDB after choosingthe appropriate ri, rd, and corresponding. The graph takes into account the combined TPS of the three services and exhibits a noticeable performance improvement in both scalability scenarios, with a greater improvement when the VM is added to poolI. Particularly, the system demonstrated good responsivenessand a consistent upward trend as demands on the computing infrastructure rose. Generally speaking, these experiments can be utilized to create a best practice strategy for a competitive situation with limitedresources, allowing scaling in an ideal manner based on the environment under consideration (see Section IV-E for more details).

C) Discussion

The first outcome of our tests is one that would seem to be obvious: the more resources devoted to a single layer, the better the performance. This incremental behavior is noticeable at both the infrastructure and database layers, where the best TPS performance is frequently attained by deploying 4 virtual machines,each of which installs a Tomcat server or a MongoDB shard, as appropriate. The database scalability test results also demonstrate MongoDB's capacity to perform better when read operations predominate over update operations. Operations outnumber updates in terms of importance. Additionally, based on the earlier findings, it can be assumed that situations in which the infrastructure and database layers both demand onthe same pool of resources will result in the same performance improvement. Here, we suppose that as the design scales out, a single VM will be added to the pool of resources, containing the resources of one VM with a Tomcat server and one VM with a MongoDB shard. Furthermore, we note that the best performance boost is seen when a competitive scaling is involved. higher priority should be given to the relevant rules when the VM is added to pool I.

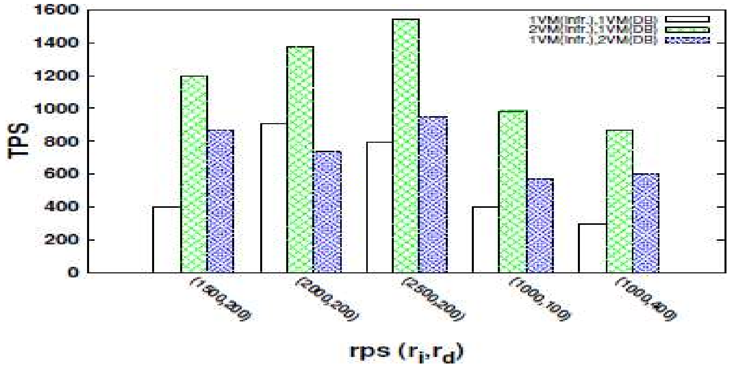

Figure 4. Competitive scaling performance

To sum up, we first note that neither our solution's primary goal was to increase the architecture's overall performance nor was it designed to offer yet another performance assessment in a cloud environment. Instead, we wanted to research how scalability affected different cloud stack levels in a competitive environment with limited resources. Our findings may therefore be viewed as a practice run that will enable the development of a performance lookup table that will direct the operations of a probe that implements a scalability manager. The probe can then employ an ideal scaling method that takes into account the unique characteristics of each user's environment (e.g., application types(.

This is based on how the user's applications are distributed in relation to the request load. The probe can alsoboost the transparency of the cloud by enabling the cloud provider to define precise and efficient scalability techniques while also providing a clear picture of how a scalability process is controlled by the cloud provider and tailored to the needs of the user.

V. RELATED WORK

Different layers can be established for scalability approaches for distributed systems and environments, which often rely on metric-based regulations [11], [12] . Traditional methods of scalability and elasticity compare the performance of various technologies that provide functionality for each cloud layer, taking into account only one cloud layer and one scenario at a time [3], [4], [13]. For instance, several strategies have concentrated on the scalability of IaaS, PaaS, and SaaS. Salah and Boutaba [15] offer a model for evaluating elastic cloud applications by predicting service response time, whereas Iosup et al. [14] analyze the performance of cloud computing services for scientific computing workloads. Espadas et al[16] .'s strategy based on resource allocation for SaaS to establish a cost-effective scalable environment offers a systematic measurement for under and over provisioning of cloud resources. Other works have analyzed the process of adaptive resource expansion and contraction, provided strategies for their performance evaluation, and controlled the flexibility of NoSQL databases and storage [3,[5], [6], [17], [18].]. Copil et al. [19] provide a multi-layer solution for the management of elasticity and scalability on the cloud, which is very similar to the work in this paper. While we focus on the issue of competing requests for resources, they define a mechanism for managing conflicting elasticity requirements.

Finally, numerous concerns have been discussed in literature in relation to cloud performance optimization. The issue of resource and data allocation has been the focus of some methodologies [20], [21]. Yi et al[21]
.'s alternative perspective on the SaaS scalability issue includes a heuristic that, given a fixed number of nodes, distributes tenants so as to maximize the sum of their numbers. DejaVu is a framework presented by Vasic et al. [20] that enhances and accelerates resource allocation in virtualized environments and can adjust to changing workloads. The capacity of a cloud framework to adapt to various circumstances has also been intensively studied in the literature, with measurements and benchmarks defined [22]–[24]. Additionally, Ali-Eldin et al. [25] describe an adaptive controller for cloud infrastructures allowing horizontal elasticity that is both proactive and reactive.

VI. CONCLUSIONS

One significant possibility provided by cloud platforms is integrated, multi-layer scalability. In this article, we discussed a method for automatic scaling in a multi-layer scenario where various cloud stack levels compete for resources that are inherently scarce. In particular, the suggested solution takes into account scalability at the database and computing infrastructure layers and is based on performance indicators and scalability criteria. In a data-intensive environment where the database and computational infrastructure layers each have a separate pool of virtual machines (VMs) and may compete for a finite number of resources, we also experimentally examined scalability criteria.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia,
‒Above the clouds: A berkeley view of cloud computing,‖ in *Tech. Rep. UCB/EECS-2009-28*, EECS Department, U.C. Berkeley, February 2009.

[2] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, National Institute of Standards and Technology (NIST), July 2009, http://thecloudtutorial.com/ nistcloudcomputingdefinition.html, Accessed January 2014.

[3] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, ―Benchmarking cloud serving systems with ycsb,‖ in *Proc. of ACM SoCC 2010*, Indianapolis, IN, USA, March 2010.

[4] CloudSpectator, *Cloud Computing Performance a Comparative Analysis of 5 Large Cloud IaaS Providers*, June 2013, http://www.scribd.com/doc/146167581/ Cloud-Computing-Performance-a-Comparative-Analysis-of- 5-Large-Cloud-IaaS-Providers, Accessed January 2014.

[10] C. Ardagna, E. Damiani, K. Sagbo, and F. Frati, ‒Zero- knowledge evaluation of service performance based on sim- ulation,‖ in *Proc. of HASE 2014*, Miami, FL, USA, January 2014, short paper.

[11] J. Caceres, L. Vaquero, A. P. L. Rodero-Merino, and J. Hierro, ―Service scalability over the cloud,‖ in *Handbook of Cloud Computing*, B. Furht and A. Escalante, Eds. Springer, 2010.

[12] R. Jimenez-Peris, M. Patiño-Martinez, B. Kemme, F. Perez-Sorrosal, and D. Serrano, ‒A system of architectural patterns for scalable, consistent and highly available multi-tier service- oriented infrastructures,‖ in *Architecting Dependable Systems VI*, R. de Lemos, J.-C. Fabre, C. Gacek, F. Gadducci, and M. ter Beek, Eds. Springer Verlag, 2009, vol. LNCS 5835.

[13] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, ‒Dynami- cally

[5] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris, ‒Automated, elastic resource provisioning for nosql clusters using tiramola,‖ in *Proc. of ACM CCGrid 2013*,Delft, The Netherlands, May 2013.

[6] T. Dory, B. Mejias, P. V. Roy, and N.-L. Tran, *Com- parative elasticity and scalability measurements of cloud databases*, http://www.nosqlbenchmarking.com/wp-content/ uploads/2011/05/paper.pdf, Accessed January 2014.

[7] A. Li, X. Yang, S. Kandula, and M. Zhang, ‒Cloudcmp: Comparing public cloud providers,‖ in *Proc. of ACM IMC 2010*, Melbourne, Australia, November 2010.

[8] C. Ardagna, E. Damiani, F. Frati, D. Rebeccani, and M. Ughetti, ‒Scalability patterns for platform-as-a-service,‖ in *Proc. of IEEE CLOUD 2012*, Honolulu, HI, USA, June 2012.

[9] J. Ekanayake, S. Pallickara, and G. Fox, ‒Mapreduce for data intensive scientific analyses,‖ in *Proc. of IEEE International Conference on eScience*, Indianapolis, IN, USA, December 2008.

scaling applications in the cloud,‖ *CCR*, vol. 41, no. 1, pp. 45–52, January 2011.

[14] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, ‒Performance analysis of cloud computing services for many-tasks scientific computing,‖ *IEEE TDPS*, vol. 22, pp. 931–945, June 2011.

[15] K. Salah and R. Boutaba, ‒Estimating service response time for elastic cloud applications,‖ in *Proc. of IEEE CLOUDNET 2012*, Paris, France, November 2012.

[16] J. Espadas, A. Molina, G. Jimnez, M. Molina, R. Ramrez, and D. Concha, ―A tenant-based resource allocation model for scaling software- as-a-service applications over cloud comput-ing infrastructures,‖ *FGCS*, vol. 29, no. 1, pp. 273–286, 2013.

[17] Y. Li and S. Manoharan, ―A performance comparison of sql and nosql databases,‖ in *Proc. of IEEE PACRIM 2013*, Victoria, Canada, August 2013.

[18] I. Konstantinou, E. Angelou, C. Boumpouka, D. Tsoumakos, and N. Koziris, ―On the elasticity of nosql databases over cloud management platforms,‖ in *Proc. of CIKM 2011*, Glas-gow, Scotland, October 2011.

[19] G. Copil, D. Moldovan, H.-L.Truong, and S. Dustdar, ―Multi- level elasticity control of cloud services,‖ in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds. Springer Berlin Heidelberg, 2013, vol. 8274, pp. 429–436.

[20] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bian- chini, ―Dejavu: Accelerating resource allocation in virtualized environments,‖ in *Proc. of ASPLOS XVII*, London, UK, March2012.

[21] Y. Zhang, Z. Wang, B. Gao, C. Guo, W. Sun, and X. Li, ―An effective heuristic for on-line tenant placement problem in SaaS,‖ in *Proc. of IEEE ICWS 2010*, Miami, FL, USA,July 2010.

[22] R. Krebs, A. Wert, and S. Kounev, ―Multi-tenancy perfor- mance benchmark for web application platforms,‖ in *Web En- gineering*, ser. Lecture Notes in Computer Science, F. Daniel, P. Dolog, and Q. Li, Eds. Springer Berlin Heidelberg, 2013, vol. 7977, pp. 424 – 438.

[23] Z. Li, L. O'Brien, H. Zhang, and R. Cai, ―On a catalogue of metrics for evaluating commercial cloud services,‖ in *Proc. of ACM/IEEE GRID 2012*, Beijing, China, September 2012.

[24] S. Islam, K. Lee, A. Fekete, and A. Liu, ―How a consumer can measure elasticity for cloud platforms,‖ in *Proc. of ACM/SPEC ICPE 2012*, Boston, MA, USA, April 2012.

[25] A. Ali-Eldin, J. Tordsson, and E. Elmroth, ―An adaptive hybrid elasticity controller for cloud infrastructures,‖ in Proc. of IEEE NOMS 2012, Maui, HI, USA, April 2012.

[26] Claudio A. Ardagna, Ernesto Damiani, Fulvio Frati, Davide Rebeccani, ―An A Competitive Scalability Approach for Cloud Architectures‖ 2014 IEEE International Conference on Cloud Computing.