Dynamically Scaling Applications in the Cloud

Luis M. Vaquero
Hewlett Packard Labs, Bristol, UK, EU
lmvaquero@ieee.org

Luis Rodero-Merino
LIP ENS Lyon, Graal/Avalon Group, INRIA, France, EU
luis.rodero-merino@ens-lyon.fr

Rajkumar Buyya
The University of Melbourne, Australia
raj@csse.unimelb.edu.au

ABSTRACT

Scalability is said to be one of the major advantages brought by the cloud paradigm and, more specifically, the one that makes it different to an "advanced outsourcing" solution. However, there are some important pending issues before making the dreamed automated scaling for applications come true. In this paper, the most notable initiatives towards whole application scalability in cloud environments are presented. We present relevant efforts at the edge of state of the art technology, providing an encompassing overview of the trends they each follow. We also highlight pending challenges that will likely be addressed in new research efforts and present an ideal scalable cloud system.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: reliability availability and serviceability, design studies

General Terms

Management, Performance

Keywords

Cloud Computing, Scalability

1. INTRODUCTION

Cloud computing is commonly associated to offering of new mechanisms for infrastructure provisioning [1, 2]. The illusion of a virtually infinite computing infrastructure, the employment of advanced billing mechanisms allowing for a pay-per-use model on shared multitenant resources, the simplified programming mechanisms (platform), etc. are some of the most relevant features.

Among these features/challenges, those introduced by adding scalability and automated on-demand self-service are responsible for making any particular service something more than "just an outsourced service with a prettier marketing face" [3].

As a result of its relevance, the wealth of systems dealing with "cloud application scalability" is slowly gaining weight in the available literature [4, 5, 6, 7, 8, 9, 10, 11, 12, 13]. As can be observed in the previous references, automation is typically achieved by using a set of service provider-defined rules that govern how the service scales up or down to adapt to a variable load. These rules are themselves composed of a condition, which, when met, triggers some actions on the infrastructure or platform. The degree of automation, abstraction for the user (service provider) and customization of the rules governing the service vary. Some systems offer users the chance of building rather simple conditions based on fixed infrastructure/platform metrics (e.g. CPU, memory, etc.), while others employ server-level metrics (e.g. cost to benefit ratio) and allow for more complex conditions (e.g. arithmetic and logic combinations of simple rules) to be included in the rules. Regarding the subsequent actions launched when the conditions are met, available efforts focus on service horizontal scaling (i.e. adding new server replicas and load balancers to distribute load among all available replicas) or vertical scaling (on-the-fly changing of the assigned resources to an already running instance, for instance, letting more physical CPU to a running virtual machine (VM)). Unfortunately, the most common operating systems do not support on-the-fly (without rebooting) changes on the available CPU or memory to support this "vertical scaling". Some authors tried to mimic this by adding more powerful servers to replace less powerful ones [11].

Beyond mere server scalability, some other elements need to be taken into account that affect the overall application scaling potential. For instance, load balancers (LBs) need to support the aggregation of new servers (typically, but not necessarily, in the form of new VMs) in order to distribute load among several servers [14, 15, 16]. Amazon already provides strategies for load balancing your replicated VMs via its Elastic Load Balancing capabilities [9]. LBs and the algorithms that distribute load to different servers are, thus, essential elements in achieving application scalability in the cloud.

Having several servers and the mechanisms to distribute load among them is a definitive step towards scaling a cloud application. However, there is another element of the datacenter infrastructure to be considered towards complete application scalability. Network scalability is an often neglected element that should also be considered [17]. In a consolidated datacenter scenario, several VMs share the same network, potentially producing a huge increase in the required bandwidth (potentially collapsing the network). It is, thus, necessary to extend infrastructure clouds to other kinds of underlying resources beyond servers, LBs and storage. Cloud applications should be able to request not only virtual servers at multiple points in the network, but also bandwidth-provisioned network pipes and other network resources to interconnect them (Network as a Service, NaaS) [17].

Clouds that offer simple virtual hardware infrastructure such as VMs and networks are usually denoted Infrastruc-

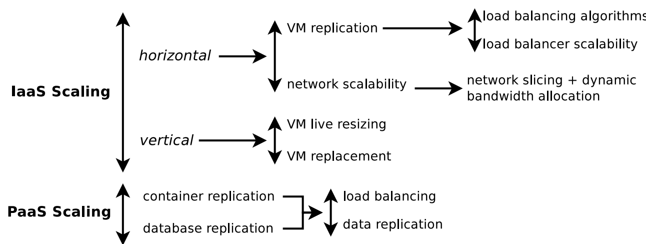

Figure 1: Summary of the Available Mechanisms for Holistic Application Scalability.

ture as a service Clouds (IaaS) [2, 18]. A different abstraction level is given by Platform as a Service (PaaS) clouds. PaaS clouds supply a container-like environment where users deploy their applications as software components [19]. PaaS clouds, provide sets of "online libraries" and services for supporting application design, implementation and maintenance. Despite being somewhat less flexible than IaaS clouds, PaaS clouds are becoming important elements for building applications in a faster manner [2, 1] and many important IT players such as Google and Microsoft have developed new PaaS clouds systems such as Google App Engine[1] (GAE) and Microsoft Azure[2]. Due to their importance this document also discusses scalability in PaaS clouds at two different levels: container level and database level.

Figure 1 provides an overview of the mechanisms handy to accomplish the goal of whole application scalability. The rest of this paper is organized as follows: Section 2 highlights state of the art technologies for managing applications in a holistic manner, including LB approaches for cloud applications. After this, Section 3 presents the few proposals dealing with the concept of NaaS. Section 4 presents current status on platform scalability and challenges ahead for the next generation of clouds. Finally, we wrap up our conclusions in Section 5.

2. SERVER SCALABILITY

Most of the available IaaS clouds deal with single VM management primitives (e.g. elements for adding/removing VMs) [4, 5, 6, 7, 8], lacking mechanisms for treating applications as a whole single entity and dealing with the relations among different application components; for instance, relationships between VMs are often not considered, ordered deployment of VMs containing software for different tiers of an application is not automated (e.g. the database's IP is only known at deployment time; thus, the database needs to be deployed first in order to get its IP and configure the Web server connecting to it), etc. Application providers typically want to deal with their application only [11, 20], being released from the burden of dealing with (virtual) infrastructure terms.

2.1 Towards Increased Abstraction and Automation: The Elasticity Controller

Such a fine-grained management (VM-based) may come in handy for few services or domestic users, but it may become intractable with a big number of deployed applications composed of several VMs each. The problem gets worse if application providers aim at having their application automatically scaled according to load. They would need to monitor every VM for every application in the cloud and make decisions on wether or not every VM should be scaled, its LB re-configured, its network resources resized, etc. Two different approaches are possible: 1) increasing the abstraction level of the provided APIs and/or 2) advancing towards a higher automation degree.

Automated scaling features are being included by some vendors [4, 9], but the rules and policies they allow to express still deal with individual VMs only; one cannot easily relate the scaling of the VMs at tier-1 with its load balancers or the scaling of VMs at tier-3, for instance. As an example of this behavior, Marshall et al. proposed a resource manager built on top of the Nimbus toolkit that dynamically adapted to a variety of job submission patterns increasing the processing power up to 10 times by federating on top of Amazon's (deploying new VMs adhered to the cluster on separate infrastructures) [13].

On the other hand, [20, 12, 11] propose more abstract frameworks (they allow users to deal with applications as a whole, rather than per individual VM) that also convey automation. Unavoidably, any "scalability management system" (or elasticity controller in Figure 2) is bound to the underlying cloud API (the problem of "discrete actuators" as named by Lim et al. [12]). One essential task for any application-level elasticity controller is, thus, mapping user scaling policies from the appropriate level of abstraction for the user to the actual mechanisms provided by IaaS clouds (depending on the specific underlying API)[3].

The implementation of the elasticity controller can be done in several different ways with regard to the provided abstraction level:

- a per-tier controller, so that there is a need for coordination and synchronization among multiple controller-actuator pairs [12]. Treating each tier as an independent actuator with its own control policy can cause shifting of the performance bottleneck between tiers. Lim et al propose that a tier can only release resources when an interlock is not being held by the other tiers.

- a single controller for the whole application (for all tiers), which let users specify how an application should scale in a global manner. For instance, the application provider could specify (based on her accurate knowledge of her application) to scale the application logic tier whenever the number of incoming requests at the web tier is beyond a given threshold [11].

2.2 Expressing How and When to Scale: Feeding Controller with Rules and Policies

All the works above rely on traditional control theory in which several sensors feed a decision making module (elasticity controller) with data to operate on an actuator (cloud API), as shown in Figure 2. Similarly, all the systems above answer the question on how to automate scalability (i.e. how to implement the elasticity controller) in a similar manner either for VM-level [4, 9] or for application-level "scalability management systems" [12, 11].

[1] `http://code.google.com/appengine`
[2] `http://www.microsoft.com/windowsazure`

[3] In practice, most of the available cloud APIs expose VM level management capabilities, so that controller-actuator pairs are limited to adding/removing VMs

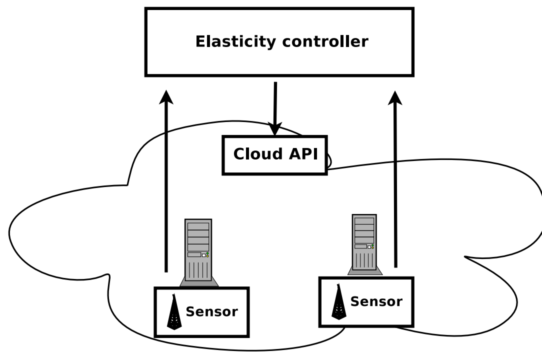

Figure 2: Conventional Control Theory Applied to the Application Level Scalability in the Cloud.

User-defined rules are the chosen mechanism (as opposed to preconfigured equations sets) to express the policies controlling scalability as shown below. A rule is composed of a series of conditions that when met trigger some actions over the underlying cloud. Every condition itself is composed of a series of metrics or events that may be present in the system, such as "money available" or "authorization received" (either provided by the infrastructure or obtained by the application provider herself), which are used together with a modifier (either a comparison against a threshold, the presence of such events) to trigger a set of cloud-provided actions.

```
RULE:

if CONDITION(s) then ACTION(s)

CONDITION:

(1..*) (metric.value MODIFIER)

ACTION:

(*) IaaS cloud-enabled actions (e.g. deploy new VM)
```

If we focus on application-level scalability (rather than dealing with per VM scaling), Lim et al. present a mathematical formulation in which the user just configures the threshold[4] for replicating storage servers so as to increase the cloud storage capability [21, 12]. Rodero-Merino et al. leave full control for users to express their rules and use a rule engine as a controller. The Open Virtual Format (OVF) is extended to define the application, its components, its contextualization needs, and the rules for scaling [10]. This way, service providers can generate a description of their application components, the way their application behaves with regards to scalability and the relevant metrics that will trigger the actions expressed in such rules.

As shown in Figure 1, in addition to dynamically adding more VM replicas, application behavior could also include many other aspects determining application scalability: adding load balancers to distribute load among VM replicas is also an important point. In an IaaS cloud the number of VMs balanced by a single LB can hugely increase, thus overloading the balancer[5].

LB scalability requires the total time taken to forward each request to the corresponding server to be negligible for small loads and should grow no faster than O(p) (p being the number of balanced VMs) when the load is big and the number of balanced VMs is large [22]. However, although mechanisms to scale the load balancing tier would benefit any cloud system, they are missing in most current cloud implementations.

Amazon already providers its Elastic Load Balancer service aimed at delivering users with a single LB to distribute load among VMs. However, Amazon does not provide mechanisms to scale LBs themselves. Virtualization of the load balancing machines offers the chance to define scalability rules by using previously presented systems [20, 12, 11]. Recently, Liu and Wee [23, 24] proposed a "rule of thumb" procedure to configure presentation-tier (Web server) scalability: for CPU-intensive web applications, it is better to use a LB to split computation among many instances. For network intensive applications, it may be better to use a CPU-powerful standalone instance to maximize the network throughput. Yet, for even more network intensive applications, it may be necessary to use DNS load balancing to get around a single instance bandwidth limitation [23]. The question emerges whether DNS-based load balancing can be scalable enough or not. Practical experiences with large well-known services such as Google's search engine and recent experimental works on cloud settings [23, 24] seem to point in that direction. Also, for many enterprise applications, hardware LB is the most common approach.

3. SCALING THE NETWORK

Properly replicated/sized VMs led us to think about LBs as a possible bottleneck. Assuming this problem is also resolved takes us to think about the link that keeps application elements stuck together, even across different cloud infrastructure services: the network, which is often overlooked in cloud computing.

Networking over virtualized resources is typically done in two different manners: "Ethernet virtualization" and overlay networks and TCP/IP virtualization. These techniques are respectively focused in the usage of virtual local area network (VLAN) tags (L2) to separate traffic or public key infrastructures to build L2/L3 overlays [25, 26, 17, 27].

Separating users' traffic is not enough for reaching complete application scalability: the need to scale the very network arises in consolidated datacenters hosting several VMs per physical machine. This scalability is often achieved by over-provisioning the resources to suit this increased demand. This approach is expensive and induces network instability while the infrastructure is being updated. Also, it is static and does not take into account that not all the applications consume all the required bandwidth during all the time. Improved mechanisms taking into account actual network usage are required. On the one hand, one could periodically measure actual network usage per application and let applications momentarily use other applications' allocated

[4]Lim et al propose the proportional thresholding mechanism, which considers the fact that going from 1 to 2 machines can increase capacity by 100% but going from 100 to 101 machines increases capacity by no more than 1%.

[5]Although we recognize the importance of load balancing algorithms, the wealth of available literature on this matter places them well beyond of the scope of this work.

bandwidth. On the other hand, applications could request more bandwidth on demand over the same links [17]. Baldine et al. proposed to "instantiate" bandwidth-provisioned network resources together with the VMs composing the service across several cloud providers [17]. Similar to the OVF extensions mentioned above, these authors employ Network Description Language (NDL)-based ontologies for expressing the required network characteristics. These abstract requirements are mapped to the concrete underlying network peculiarities (e.g. dynamically provisioned circuits vs. IP overlays). A complete architecture to perform this mapping has also been proposed [28]. Unfortunately, there is no known production-ready system that fully accomplishes the need for dynamically managing the network in synchrony with VMs provisioning.

These techniques to increase the utilization of the network by virtually "slicing" it have been dubbed as "network as a service" [17]. This *à la cloud* network provision paradigm can be supported by flow control [29], distributed rate limiting [30], and network slicing techniques [31]. By applying this mechanism the actual bandwidth can be dynamically allocated to applications on demand, which would benefit from a dynamic resource allocation scheme in which all the users pay for the actual bandwidth consumption. To optimize network usage statistical multiplexing is used to compute the final bandwidth allocated to each application. Statistical multiplexing helps to allocate more bandwidth to some applications while some others are not using it (most system administrators usually provision on a worst-case scenario and never use all the requested resources). This way, the cloud provider can make a more rational use of its network resources, while still meeting applications' needs.

4. SCALING THE PLATFORM

IaaS clouds are handy for application providers to control the resources used by their systems. However, IaaS clouds demand application developers or system administrators to install and configure all the software stack the application components need. In contrast, PaaS clouds offer a ready to use execution environment, along with convenient services, for applications. Hence, when using PaaS clouds developers can focus on programming their components rather than on setting up the environment those components require. But as PaaS clouds can be subject to an extensive usage (many concurrent users calling to the hosted applications), PaaS providers must be able to scale the execution environment accordingly. In this section, we will explore how scalability impacts on the two core layers of PaaS platforms: the *container* and the *database management system* (DBMS), as they are the backbone of any PaaS platform: the combination container + database is the chosen stack to implement many networked (e.g. Internet) applications, which are the ones PaaS platforms are oriented to.

The container is the software platform where users' components will be deployed and run. Different PaaS clouds can be based on different platforms, for example GAE and its open source counterpart AppEngine [32] provide containers for servlets (part of the J2EE specification) and Python scripts, while Azure and Aneka [33] offer an environment for .NET applications. Each platform type can define different lifecycles, services and APIs for the components it hosts.

Databases, on the other hand, provide data persistence support. The database storage service must address the demand for data transactions support combined with big availability and scalability requirements. As it is explained later in this section, this can be addressed in different manners.

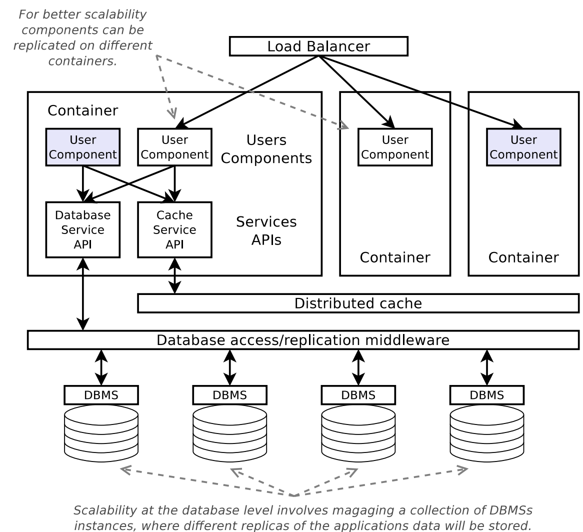

Figure 3: Schematic View of a PaaS System.

Figure 3 shows an overview of the possible architecture of a PaaS system, where both the container and the database layer achieve scalability through replication of the container and the DBMS (horizontal scaling). This is the scenario this work focuses on, as it is the only one the authors deem feasible in clouds with certain scalability requirements. Applying vertical scaling by using "more powerful" hardware would soon fail, as many clouds will typically face loads that one single machine cannot handle whatever its capacity.

Other services can be offered by a PaaS platform apart from the container and the database, which also will need to be scaled to adapt to demand. For example, Azure offers services for inter-component communication (bus) or access control. Unfortunately, in this work it would only be possible to study scalability issues of all those services in a too superficial manner. Instead, a most thorough analysis of the most important services, the container and the database, has been preferred.

4.1 Container-level Scalability

At container level, a better scalability can be achieved by enabling multitenant containers (having the ability to run components belonging to different users). This imposes strong isolation requirements which maybe not all platforms can achieve by default. For example, the standard Java platform is known to carry important security limitations that hinder the implementation of safe multitenant environments [34]. A more straightforward option is to run each user's components on non-shared containers, which is the approach taken by GAE.

In both cases scaling is implemented by instantiating/releasing containers where several replicas of the developer components can be run (i.e. it is a form of horizontal scaling). This should automatically be done by the platform, so developers are not forced to constantly monitor their services' state. Either automatic scaling IaaS systems (such as those mentioned in Section 2) can be used, or the platform itself

can scale up and down the resources. The latter approach is the one used by both AppEngine and Aneka. AppEngine can run in any Xen based cloud, private (built for example with Eucalyptus [35]) or public (such as EC2). However, it is not clear from [32] if AppEngine supports transparent cloud federation so that a private PaaS cloud could deploy its containers in VMs running in third party-owned IaaS clouds to face sudden load peaks. In any case AppScale could apply Eucalyptus ability to be integrated with other clouds. Aneka, on the other hand, supports cloud federation natively, so federated Aneka clouds can borrow resources among them, or a given Aneka cloud can use containers hosted in VMs running in different IaaS clouds.

Automatic scaling of containers has several implications for developers regarding component design. If the PaaS platform can stop container replicas at any moment (for example due to low load), components could be designed to be as stateless as possible (as GAE recommends for the applications it hosts). In order to ease application development, support for stateful components should be offered by the platform. In this case with stateful components, LB and container replica management modules must be aware of state. This way, LBs know which container replicas hold session data to forward requests accordingly; and container instances are not removed as long as they hold some session.

If more than one container instance holds data from the same session (for better scalability and failure resilience), then some mechanism is necessary that allows to keep the different data replicas updated. Transparent session data (e.g. "shopping cart") replication, usually denoted *soft state replication* can be offered through systems such as Tempest [36] tool or SSM [37]. It is also possible to use distributed cache systems such as memcached [38] for explicit data sharing among component replicas. Each solution will have a different impact on component development. Roughly speaking, distributed caches work at application level, i.e. they are explicitly accessed by the hosted components code to store/retrieve the information shared among component replicas, while soft state/session replication systems work in a transparent manner for the application developer.

4.2 Database Scalability

The abundance of literature on database scalability is huge, but only the most important points for PaaS databases are highlighted here. PaaS systems must expect very high requests rates as they can host many applications that demand intense data traffic. Three mechanisms can be used (and combined) to handle this: distributed caching, NoSQL databases, and database clustering.

Caching systems, such as memcached [38], are used to store intermediate data to speed up frequent data queries. A request for some data item will first check the cache to see if the data is present, and will query the database only if the data is not in the cache (or it has expired). Distributed caching systems provide the same cache across several nodes, which is useful in clustered applications. For example, GAE offers memcache[6] as a service for application developers.

The term "*NoSQL*" refers to a wide family of storage solutions for structured data that are different from the traditional relational, fully SQL-compliant, databases [39]. NoSQL systems offer high scalability and availability, which seems a good fit in cloud environments with potentially many applications hosted under high demand. On the other hand, the replica management mechanisms they use, provide less guarantees than traditional systems. Usually, updates on data copies are not immediately done after each write operation, they will be "eventually" done at some point in the future. This causes these systems to be unable to implement support for transparent and fully ACID compliant transactions, hence imposing some limitations on how transactions can be used by developers. Besides, the fact that they only support (the equivalent to) a subset of SQL can be a hurdle for some applications. An example is BigTable [40], which is used by GAE to provide its object oriented data storage service. HBase[7], an open source implementation of BigTable, is used by AppEngine.

Finally, if fully relational and SQL-compliant databases are to be provided (as in the case of Microsoft's Azure), clusters can be built to provide better scalability, availability and fault tolerance to typical DBMS systems. Unfortunately, these clusters must be built so several or all nodes contain a replica of each data item, which is known to compromise performance even for moderate loads when transactions are supported [41]. Present database replication systems from every major relational DBMS have several limitations, and further research is needed to achieve the desired performance [42]. The major problem comes from the fact that transactions require protecting the data involved while the transaction lasts, often making that data unavailable to other transactions. The more transactions running at the same time, the more conflicts will be raised with the corresponding impact on the application performance.

Yet, some database replication solutions exist that offer some degree of scalability. These can be part of the DBMS itself (in-core) or be implemented as a middleware layer between the database and the application. Most of the middleware based solutions use a proxy driver used by client applications to access the database. This proxy redirects requests to the replication middleware, which forwards them to the database. The middleware layer handles requests and transforms then in database operations to ensure that all data copies are updated. Also, it takes care of load balancing tasks. Examples of such solution are C-JDBC [43], Middle-R [44] and DBFarm [45].

It can be concluded from this section that replication of databases/components is the most important issue to consider when scaling a PaaS platform. Accordingly, Figure 4 sketches the main replication ideas presented in this section. At container level, the same component can be run on different container instances to achieve better scalability and failure tolerance. But then the platform should make available some mechanism for consistent data replication among components. At database level, copies of the application data can be hosted in different DBMS instances again for better scalability and failure tolerance. Unfortunately, keeping consistency can lead to transaction conflicts.

Table 1 sums up the most relevant works related to holistic application scaling in cloud environments at the three different levels discussed in this short review: server, network and platform level. The most relevant features are highlighted and appropriate references are given for readers' convenience.

[6] http://code.google.com/appengine/docs/java/memcache/

[7] http://hbase.apache.org

SERVER LEVEL	
Automatic VM Scaling [4, 9]	Services that scale single VMs horizontally depending on a set of predefined, fixed and VM-related performance metrics.
Dynamic Workload-pattern Matching [13]	Nimbus scaled out by federating on top of Amazon's (deploying new VMs adhered to the cluster on separate infrastructures as in the row above). However, Nimbus included a new technique: it dynamically adapted to a variety of job submission patterns, which resulted in further scalability.
Whole Application Scaling [20, 12, 11]	Mechanisms to express the scaling features of the whole service are provided by these systems. Complex rules are available based on service performance metrics that relate measurements of different VMs or different tiers to control scalability.
Non-scalable Load Balancing	Amazon offers Load Balancers to distribute load among your created VM replicas. However, this system does not offer any mechanism to scale load balancer themselves.
DNS-based Load Balancing	DNS load balancing seems to be a reasonable approach in a public cloud where every VM receives a public IP. What is the away to go in private or hybrid clouds in which application components can be placed in public and private clouds.
NETWORK LEVEL	
On-demand Creation of Virtual Network Resources [28, 17]	An architecture and proof of concept system are available that "instantiate" bandwidth-provisioned network resources together with the VMs composing the service across several cloud providers.
Network slicing [29, 17, 31, 30]	Keep separate per application flows by adapting to on demand network utilization needs by every application, dynamic network bandwidth allocation.
PLATFORM LEVEL	
AppScale [32]	Platforms will require container replicas to be deployed or released dynamically to handle load variations. AppScale can scale the VMs used to host containers depending on actual application demand, automatically configuring the load balancers.
Aneka [33]	For high loads, and to avoid overprovisioning of resources, it would be useful to be able to federate clouds so components can be run in external/public clouds if needed. Aneka is able to deploy containers and run users applications in several IaaS providers.
Tempest/SSM [36]	"Soft state" in its title refers to data that does not need to be permanently stored, such as user session data. Replication of soft state data makes such data available to all application replicas so that everyone can attend user requests.
Automatic Session Replication	Some container implementations can use soft state replication solutions or their own replication system for automatic replication of users sessions. These solutions work at container level and are transparent to the application developer.
memcached [38]	Distributed cache systems, such as memcached, offer a key/value distributed storage system that can be used to reduce database access requests. The values stored are available to all application replicas, so distributed caches can be used to share state information among those replicas in an explicit manner.
BigTable/HBase [40]	Traditional fully SQL and ACID compliant DBMSs have limited scalability. Recent DBMSs are rather oriented to high availability and scalability, although they can relax some ACID conditions and do not fully implement SQL. This approach can be more suitable in cloud platforms.
C-JDBC/Middle-R/DBFarm [43, 44, 45]	If fully SQL and ACID compliance is a requirement, then an option to increase the scalability is to combine several DBMS to manage replicas of all or part of each database. C-JDBC and Middle-R provide a middleware layer that allows to combine DBMS in a flexible manner.

Table 1: Works Related with Scalability at Different Levels

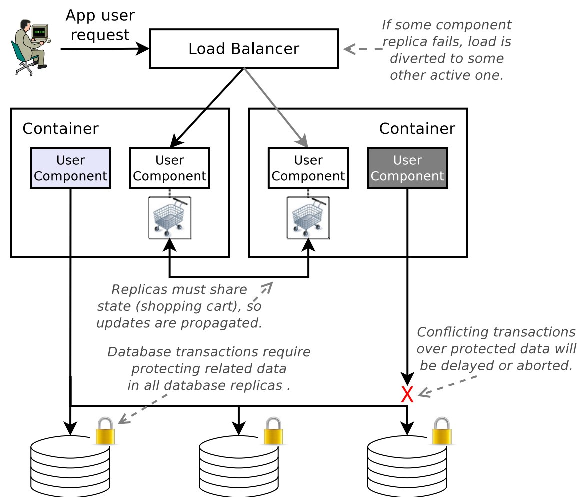

Figure 4: Data Replication in PaaS.

5. CONCLUSIONS

There have been great advances towards automatically managing collections of interrelated and context-dependent VMs (i.e. a service) in a holistic manner by using policies and rules. The degree of resource management, the bonding to the underlying API and coordinating resources spread across several clouds in a seamless manner while maintaining the performance objectives are major concerns that deserve further study. Also, dynamically scaling LBs and its effects on whole application scalability are yet to be reported.

The reported works on network scalability are also scarce. Some experimental reports have shed some light on possible ways to go, but there is no known production-ready system that fully accomplishes the need for dynamically managing the network in synchrony with VM provisioning. Also, such approaches will have to convince carriers, which are very reluctant to introduce innovations in production networks since they can damage the provided service when taken to such a demanding production environment.

On the other hand, to achieve proper PaaS scalability cloud providers must address issues both at container and database level. Multitenant containers could be used to save resources, but this implies unsolved security concerns, while non-shared containers will demand more resources (and so imply more operation costs for the cloud provider). Replication of components and databases can be also applied for better scalability. However, replication often brings strong performance penalties that must be taken into account. How to achieve replication without incurring in high performance degradation is an open research topic.

Grouping all the elements reviewed in this work in different functionalities provides us with the set of must have elements in an "ideal" elastic cloud. This system should contain the appropriate elements so that applications can be scaled by replicating VMs (or application containers), by reconfiguring them on the fly, and by adding load balancers in front of these replicas that can scale by themselves (even relying on DNS to do so). Also, the selection of the

appropriate algorithms for load balancing is a crucial decision element. At the network level, such ideal scalable cloud should offer users the chance to dynamically ask for the network resources they are actually using. For instance. enabling applications to ask for more dedicated bandwidth at a given stage in the application lifecycle, or adding virtual routers to create a separated Internet-connected overlay for an application.

Regarding PaaS, the "ideal" scalable platform should be able to instantiate or release instances of users' components as demand changes, and transparently distribute the load among them. To ease developers tasks, the concept of session should be implemented by the platform, which requires support for (transparent) data replication. But data replication requires that component instances keep an updated copy of the data. The necessary updates consume bandwidth and time, and can lead to big delays on requests processing. An equivalent problem is found at database level. A PaaS platform should ideally give access to traditional relational databases with support for ACID transactions. But as more replicas are created to attend an increasing demand, the overload necessary to keep consistency will induce delays on requests. The ideal PaaS cloud must balance the need for powerful programming abstractions that ease developers tasks with the support for transparent scalability.

6. DISCLAIMER

The opinions herein expressed do not represent the views of HP Labs or INRIA. The information in this document is provided as is, no guarantee is given that the information is fit for any particular purpose. The above companies shall have no liability for damages of any kind that may result from the use of these materials.

7. REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.

[3] D. Owens, "Securing elasticity in the cloud," *Queue*, vol. 8, no. 5, pp. 10–16, 2010.

[4] (2010, May) Righscale web site. [Online]. Available: http://www.rightscale.com

[5] (2010, May) vcloud api programming guide. [Online]. Available: http://communities.vmware.com/static/vcloudapi/vCloud_API_Programming_Guide_v0.8.pdf

[6] (2010, May) Sun cloud web site. [Online]. Available: http://kenai.com/projects/suncloudapis

[7] (2010, May) Gogrid web site. [Online]. Available: http://www.gogrid.com

[8] J. Varia. (2008, Sept) Amazon white paper on cloud architectures. [Online]. Available: http://aws.typepad.com/aws/2008/07/white-paper-on.html

[9] (2010, Aug) Amazon auto scaling service. [Online]. Available: http://aws.amazon.com/autoscaling/

[10] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero, "Service specification in cloud environments based on extensions to open standards," in *COMSWARE '09: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*. New York, NY, USA: ACM, 2009, pp. 1–12.

[11] L. Rodero-Merino, L. Vaquero, V. Gil, F. Galán, J. Fontán, R. Montero, and I. Llorente, "From infrastructure delivery to service management in clouds," *Future Generation Computer Systems*, vol. 26, pp. 1226–1240, October 2010.

[12] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *ICAC10*. New York, NY, USA: ACM, 2010, pp. 19–24.

[13] P. Marshall, K. Keahey, and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 43–52, 2010.

[14] E. Berger and J. C. Browne, "Scalable load distribution and load balancing for dynamic parallel programs," in *IWCBC99: In Proceedings of the International Workshop on Cluster-Based Computing 99, Rhodes/Greece*, 1999.

[15] S. Olivier and J. Prins, "Scalable dynamic load balancing using upc," in *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 123–131.

[16] H. Wu and B. Kemme, "A unified framework for load distribution and fault-tolerance of application servers," in *Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 178–190.

[17] I. Baldine, Y. Xin, D. Evans, C. Heerman, J. Chase, V. Marupadi, and A. Yumerefendi, "The missing link: Putting the network in networked cloud computing." in *ICVCI09: International Conference on the Virtual Computing Initiative*, 2009.

[18] L. Youseff, M. Butrico, and D. da Silva, "Toward a unified ontology of cloud computing," in *Proceedings of the Grid Computing Environments Workshop*, Austin, Texas, USA, November 2008, pp. 1–10.

[19] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? An architectural map of the cloud landscape," in *ICSE 2009: Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, Vancouver, Canada, May 2009, pp. 23–31.

[20] R. Buyya, R. Ranjan, and R. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services." in *ICA3PP 2010: The 10th International Conference on Algorithms and Architectures for Parallel Processing*, 2010, pp. 19–24.

[21] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds*. New York, NY, USA: ACM, 2009, pp. 13–18.

[22] Berger, Browne, E. Berger, and J. C. Browne, "Scalable load distribution and load balancing for dynamic parallel programs," in *In Proceedings of the*

International Workshop on Cluster-Based Computing 99, Rhodes/Greece, 1999.

[23] H. Liu and S. Wee, “Web server farm in the cloud: Performance evaluation and dynamic architecture,” in *CloudCom ’09: Proceedings of the 1st International Conference on Cloud Computing*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 369–380.

[24] S. Wee and H. Liu, “Client-side load balancer using cloud,” in *SAC ’10: Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010, pp. 399–405.

[25] X. Jiang and D. Xu, “Violin: Virtual internetworking on overlay infrastructure,” in *PDPA03: Proceedings of the 2nd International Symposium on Parallel and Distributed Processing and Applications*, 2003, pp. 937–946.

[26] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, “In vini veritas: realistic and controlled network experimentation,” in *SIGCOMM ’06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2006, pp. 3–14.

[27] T. W. Alex, P. Shenoy, and J. V. Merwe, “The case for enterprise-ready virtual private clouds,” in *HotCloud09: Proceedings of the Workshop on Hot Topics in Cloud Computing.*, 2009, pp. 1–5.

[28] M. Keshariya and R. Hunt, “A new architecture for performance-based policy management in heterogeneous wireless networks,” in *Mobility ’08: Proceedings of the International Conference on Mobile Technology, Applications, and Systems*. New York, NY, USA: ACM, 2008, pp. 1–6.

[29] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar, “Carving research slices out of your production networks with openflow,” *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 129–130, 2010.

[30] B. Raghavan, K. Vishwanath, S. Ramabhadran, K. Yocum, and A. C. Snoeren, “Cloud control with distributed rate limiting,” in *SIGCOMM ’07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2007, pp. 337–348.

[31] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, “Path splicing,” in *SIGCOMM ’08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. New York, NY, USA: ACM, 2008, pp. 27–38.

[32] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski, “AppScale design and implementation,” UCBS, Tech. Rep. 2009-02, 2009.

[33] C. Vecchiola, , X. Chu, and R. Buyya, *Aneka: A Software Platform for .NET-based Cloud Computing*. IOS, 2009, pp. 267–295.

[34] A. Herzog and N. Shahmehri, *Problems Running Untrusted Services as Java Threads*, ser. IFIP International Federation for Information Processing. Springer, September 2005, vol. 177/2005, pp. 19–32.

[35] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The eucalyptus open-source cloud-computing system,” in *CCGRID’09: Proceedings of 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Shanghai, China, May 2009, pp. 124–131.

[36] T. Marian, M. Balakrishnan, K. Birman, and R. van Renesse, “Tempest: Soft state replication in the service tier,” in *DSN 2008: Proceedings of the 38th International Conference on Dependable Systems and Networks*, Anchorage, Alaska, June 2008, pp. 227–236.

[37] B. C. Ling, E. Kiciman, and A. Fox, “Session state: beyond soft state,” in *NSDI 2004: Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, San Francisco, California, USA, March 2004, pp. 295–308.

[38] J. Petrovic, “Using memcached for data distribution in industrial environment,” in *ICONS 2008: Proceedings of the 3rd International Conference on Systems*, Cancún, México, April 2008, pp. 368–372.

[39] N. Leavitt, “Will NoSQL databases live up to their promise?” *Computer*, vol. 43, pp. 12–14, February 2010.

[40] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Transactions on Computer Systems*, vol. 22, June 2008.

[41] J. Gray, P. Helland, P. O’Neil, and D. Shasha, “The dangers of replication and a solution,” in *SIGMOD 1996: Proceedings of the 16th ACM International Conference on Management of Data*, Montreal, Quebec, Canada, June 1996, pp. 173–182.

[42] E. Cecchet, G. Candea, and A. Ailamaki, “Middleware-based database replication: the gaps between theory and practice,” in *SIGMOD 2008: Proceedings of the 28th ACM International Conference on Management of Data*, Vancouver, Canada, June 2008, pp. 739–752.

[43] E. C. Julie, J. Marguerite, and W. Zwaenepoel, “C-jdbc: Flexible database clustering middleware,” in *USENIX 2004: Proceedings of the USENIX 2004 Annual Technical Conference*, June 2004, pp. 9–18.

[44] J. M. Milán-Franco, R. Jiménez-Peris, M. Patino-Pérez, and B. Kemme, “Adaptative middeware for data replication,” in *Middleware’04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, Toronto, Canada, October 2004, pp. 175–194.

[45] C. Plattner, G. Alonso, and M. T. Özsu, “DBFarm: a scalable cluster for multiple databases,” in *Middleware’06: Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, 2006, pp. 180–200.