Why Elasticity Matters

Technical Report BUCS-TR-2012-006

Dan Schatzberg
Boston University

Jonathan Appavoo
Boston University

Orran Krieger
Boston University

Eric Van Hensbergen
IBM Austin Research Lab

1 Introduction

Elasticity should be treated as a first class system parameter. Particularly in large cloud environments, elastic applications would benefit if the underlying infrastructure provided primitives for elasticity and were themselves elastic. If you want to provide an elastic service and the cloud does not provide good primitives for the degree of elasticity you require, then you are forced to over-provision – acquire more resources than you instantaneously need and subsequently hoard them. Doing so hinders the cloud's ability to optimize global system utilization. Free or idle resources become hidden. If however, each cloud layer provides appropriate primitives that permit resources to be acquired and released at a scale that is equal to or better than what is required, then hoarding is less likely to occur. This permits the cloud infrastructure to collectively migrate resources to the real demand. To achieve this in a multi-layer system, demand must be transparently reflected from top to bottom. We must focus on the design and evaluation of primitives for expressing and managing elasticity at all levels, across nodes, and potentially across data centers.

If research focuses on pushing the boundaries of elasticity, new classes of applications can be developed. For example, if a cloud would permit an application to grow and shrink the use of thousands of processors between mouse clicks, then *High Performance Interactive Applications* would be viable. Consider a medical imaging and analysis application. Using a raw megapixel image with an algorithm requiring quadratic memory in the size of the input, this requires roughly 14 terabytes of memory, putting it well outside the reach of the ram capacities of desktop computers. However, a "small" supercomputer today ($1/10$ of the largest current IBM BlueGene P System), capable of approximately 10^{14} operations per second, can not only contain the data, but can perform an operation on each data value in under a second. All of a sudden, operating on the image not only becomes viable, but we can even do it at interactive speeds.

While an interactive version of this application has large value, it is not feasible today. Suppose a doctor's office had the necessary software and wanted to use Amazon's EC2 HPC offering for an 8 hour work day. To operate on the image would require 623 compute instances[1]. Given pricing at the time of writing, this translates to approximately $\$8000.00$ per day. Due to the interactive nature of the application, the actual utilization of the instances will be a small fraction of the time that is being paid for. This is likely a cost prohibitive proposition. If, however, it was possible to acquire and release the resources at interactive time scales, then the instances could be reallocated to other EC2 users and the doctor's cost would more closely reflect the usage. Researching dramatically higher degrees of elasticity with respect to the scale of the resources and duration they are held would enable such high performance interactive applications.

If we develop effective ways of exporting the elasticity via designed and usable primitives, then we can not only ease the burden of developing elastic applications and services, but also we can foster and encourage them. We can reduce the application development burden by providing support for representing and reflecting dynamic demand and translating it into dynamic requests for resources. Similar to how a traditional operating system transparently manages memory via mappings and pages faults, one can explore how systems can enable primitives for elasticity.

In summary we argue that elasticity is an important area of research and hypothesize that research in this area will lead to more efficient systems with less hoarding, new applications that exploit massive cloud resources elastically, and system software and libraries that will simplify the task of developing elastic applications.

The paper is structured as follows: In the next section we explain our observations regarding the design space for a system that supports extreme levels of elasticity to aid highly elastic application development. In section three, we discuss our Scalable Elastic Systems Architecture (SESA) inspired by our observations and we conclude the paper in section four.

2 Elasticity and Systems

Cloud computing has changed the scale at which we think of systems. The resources of a data center and potentially even multiple data centers have become one multi-user system. A few mouse clicks can cause thousands if not hundreds of thousands of computers to consume electricity. This has changed the way we do things. It is now easy for anyone, given a big enough credit card limit, to develop and test a new peer to peer file system on thousands of computers. A start-up can be isolated, without over provisioning, at least for some period of time, from the load induced by getting slashdotted because the cloud service they use transparently absorbs the load by distributing it to more of their underlying resources.

As cloud computing enables dynamic compositional nesting of producer and consumer based services, the very notion of who and what a user is becomes increasingly vague. Applications make use of potentially multiple providers at various levels of abstraction and transparency ranging from hardware, to virtual machines, to web hosting, to development platforms, to data management, to identity management, to customer relations, etc. with the application itself finally showing up as a banner on a web page. And yet someone must actually be paying for all the resources that are consumed to do all the work. The space and energy consumed by a computer is not virtual.

While elasticity seems inherent, how should we as systems researchers try and formalize it with respect to systems architecture? In this section we define three goals of an elastic systems architecture: Demand for resources must flow top-down through the system; Support for elasticity should be expressed from the bottom-up; Modularity should be exploited to support a diverse set of applications.

2.1 Top-down Demand

The heart of elasticity is the varying demand for resources. There is a notion of external and internal demand where external demand is driven from the outside of a service and internal is an attribute of how the service is implemented. One can think of external demand as load and naturally expressed in terms of requests for service per unit of time whereas internal demand is a measure of the resources required by the service. Ideally, from an elasticity point of view, the internal demand would be a direct function of load with a very small base fixed cost that would be close to zero. In other words, a service's internal demand is a translation of the external demand into temporal and spatial requirements of resources.

With respect to layering, where application software is towards the top and hardware towards the bottom, one can view demand requirements flowing top down. The role of arbitration of access to system resources based on external demand has long been the role of the operating system. However, existing operating systems are built based on a notion of fixed underlying resources and neither have the ability to express the need for additional resources to the (potentially virtual) hardware layer below them nor the ability to actively release resources which are not currently being used. Any information about demand is communicated implicitly via access to said resources (ie. the mapping of physical memory into the page table, the use of disk blocks, etc.) without semantic information regarding how the resource is being used (whether a page is actually a currently unused block of the file system page cache). This lack of fidelity in demand information becomes acute as we traverse down the layers of a traditional cloud computing stack.

Given cloud computing environments in which a data center can contain massive amounts of resources, flowing demand down to the lowest layers would enable idle resources to migrate between the largest possible population of users. As one moves further up from the lowest level, HaaS, towards the IaaS, PaaS and SaaS layers[19] the demand by definition is a subset of the lower layer. The lower in the stack that resources can be freed allows the resources to be reallocated to the sources of demand that are aggregated by that layer.

Google App Engine[7] is an example of a fine grain PaaS interface that is designed to enable applications to be developed in an elastic manner on top of the resources assigned to Google App Engine infrastructure. Applications are provided with a web oriented transactional execution model. The load on an application in the form of web requests are converted into threads that have a fixed quantum to execute in. The App Engine infrastructure allocates as a function of requests to an application.

Snowflock[15] is an example of a more elastic IaaS interface that encourages elasticity in the use of resources in the form of virtual machines. Building on a process like execution model, it encourages developers to build services that fork VM's. Forking a VM automatically acquires additional resources of the IaaS. Similarly, when a VM exits, the resources are returned back to the IaaS. Doing so encourages demand to be mapped to virtual machine instances. VM's can be forked as a function of service level requests.

While there may be other options, elasticity seems to lead one naturally to an event driven model. One can easily map service oriented architectures to an event driven model by treating each service request as an event that is dispatched to the resources, either logical or physical. Events provide a convient way of expressing both the spatial and temporal changes in demand. They can also be associated with resources and dynamic resource

allocation and deallocation. Having each layer of the system, like the stages in SEDA[17], be event driven would allow demand as events to flow top-down from the application to the hardware.

2.2 Bottom-up support

A properly designed elastic architecture will take into account the attributes of the physical units and enable the full exploitation of a system that supports fine grained physical resource allocation with minimal latency. As cloud computing hardware evolves to incorporate tailored data center scale interconnects, we expect to see hardware units composed of processors, ram and switching capacity that can be reallocated in milliseconds[3, 14, 4]. An elastic systems architecture will be able to evolve with these machines. In particular, the hardware and the low-level allocators and hardware schedulers themselves need to be explicit in their interfaces with respect to the spatial and temporal elasticity they support.

A natural way to reflect the elasticity of the physical resources is to imploy a logical resource model for software. A software layer can be viewed as a provider of logical resource instances, ranging from virtual machines to threads and data structures. The physical units ultimately are utilized through their allocation to logical resources. As we move up from the hardware, layers should employ allocators and schedulers that also are explicit in their interfaces with respect to elasticity. In this way, through a uniform physical and logical resource model, the support for elasticity can flow bottom-up from the hardware to the applications.

2.3 Exploit Modularity

Cloud computing architectures are composed of abstract layers of distributed systems. Elasticity can benefit from a layered approach by introducing alternative elastic implementations that can co-exist with existing implementations. Modularity, in the form of distributed components and object-oriented programming, is a common paradigm for distributed software construction. Elasticity can also benifit from component modularity as each logical resource can be mapped to a component instance. Additionally, component implementations that internally exploit elasticity can be introduced and utilized on a case by case basis. Components also naturally integrate with an event driven model. Events can be implemented as a flow of execution that is directed to elastically allocated compositions of component instances, both within a layer and across layers.

Modular layering can enable tighter coupling between applications and lower layers. For instance, applications can choose to use new distributed operating system layers or bypass middleware and operating system services altogether and operate directly with the interfaces provided by the cloud infrastructure. This approach has been taken in the past at the node level by bare-metal Java virtual machines and at data-center scale by high-performance computing applications. In such scenarios it would be desirable for system abstraction layers to provide guidance and enable bypass mechanisms. This could then be used to selectively bind in implementations of interfaces which better support the elasticity facilities we have discussed in the previous two sections.

Emerging operating system kernel designs such as fos [18], Tesselation [11], Barrelfish multikernels [5], and Helios [13] are all examples of how modularity can be incorporated into the systems software layer in a dynamic and distributed fashion. The next logical step is to apply similar techniques to the virtualization and cloud infrastructure layers and expose elastic aware interfaces which seamlessly span all system layers and nodes distributed throughout the cloud substrate. Elasticity and the configurability enabled by this modularity seem synergistic. For example, a system that permits an application to customize the page fault behavior on a specific memory mapping can allow the semantics of the mapping to influence how many pages are acquired on initial faults and when the pages should be released.

Elasticity can benefit from modularity if we carefully exploit the configurability and the potentials for supporting an elastic event and resource model that it offers.

2.4 Summary

Based on our observations, we posit the following goals for a systems architecture for elasticity:

Top-Down Demand The system should enable demand on services to flow from high level layers as transparently as possible to the lowest layers of the system. Hoarding should be discouraged or at least made transparent. Event driven interfaces and services should be supported and encouraged by the system.

Bottom-Up Support We advocate that elasticity should be an explicit characteristic that should be supported and made explicit in the lowest layers of a system and, if possible, all the way into the hardware. The construction of layers that are explicit about the elasticity they provide with respect to the base elasticity of the system should be encouraged via systems support.

Exploit Modularity Use modularity to enable applications to map elasticity in their demand as closely to the capabilities of the system when desired. Embrace the layering of cloud computing as a abstract architecture but ensure that all but the lowest layer can have varying implementations or be overridden. Further, provide some form of support for

a component model that enables the base elasticity to be exploited by new and advanced applications.

3 SESA

Motivated by these observations, we are exploring a Scalable Elastic System Architecture (SESA) which aims to enable extreme degrees of elasticity across all system layers, aiding in the construction of elastic software. Along with an abstract layering, we propose a distributed elastic component model that can be used to construct layers of the system that are tuned to exploiting fine grain elasticity.

SESA defines four meta layers to a system that can easily be mapped to current and future cloud computing environments. At the bottom is the physical Elastic Node/HW Layer that represents a data center's underlying computer resources. We assume a model in which the resources of the data center are decomposed into *nodes* that form the basic unit of resource allocation and thus elasticity. The next layer up, the Elastic Partition Layer, provides groups or partitions of node resources that can be associated with a consumer or principle. A partition is the basic unit by which a principle can elastically aggregate node resources. To enable applications to scale and exploit the elasticity of a partition, the next layer provides an Elastic Building Block model. Elastic Building Blocks (EBBs) are the primary way in which application software is structured so that it is scalable and changes in demand can be converted into elastic consumption of node resources for a particular application of a principle. The top layer is the Elastic Service and Runtime Layer. The specific service and or runtime code of the application is written as a dynamically allocated set of EBBs.

Our systems software focus is on the top two layers. Our goal is to introduce a distributed library os runtime that enables Elastic Building Blocks (EBBs) for the construction of system and application layers that express and exploit the maximum elasticity of the hardware to meet the demands of interactive workloads. Our model attempts to achieve the goals of top-down demand, bottom-up support and exploiting modularity. Specifically, it uses an event model that maps events to method invocations of EBB's. EBB construction and destruction are by default to be triggered by event access and quiescence. The components will have an associated IPC like model so that invocations can cross layers. The library model will enable EBB constructed services to be constructed and deployed in conjunction with existing implementations.

4 Concluding remarks

Systems in the past were designed to efficiently share fixed resources among a mix of different applications. We are currently building our clouds and elastic cloud applications using the HW and SW systems that arose from this legacy.

In this paper we proposed a new research agenda focused on elasticity. We argued that elasticity is an important area of research and hypothesized that research in this area will lead to more efficient systems with less hoarding, new applications that exploit massive cloud resources elastically, and system software and libraries that will simplify the task of developing elastic applications.

We discussed some of our thoughts on a top-to-bottom cloud-scale system focused on elasticity. We argued that such a system will require: 1) a HW/IaaS layer that can quickly reallocated resources to different applications, 2) an event driven model where resource demand flows from the high level layers as transparently as possible to the lowest level of the system, and 3) a model of modularity that allows layers to be overridden as necessary and provides applications with a component model that enables the base elasticity to be exploited by new and advanced applications.

These requirements have led to the design of the SESA system briefly described in the previous section. We are just starting to build this system. The elastic building blocks extend our previous work on building blocks from K42[10] and draw additional inspiration from Fragmented Objects[12], Distributed Shared Objects[8], and Distributed Shared Abstractions[6]. We will incorporate EBB into a library OS inspired by our experience with Libra [2]. This library OS will be able to automatically and quickly extend across and release VMs or HW nodes as the application's demands change. Our prototype will initially focuse on a matlab like environment built utilizing SAGE[16].

We expect to rapidly have an end-to-end simple operational implementation of SESA focused on a medical imaging application on top of SAGE running on both BG/Q systems and vCloud[9]. Focusing this work on one relavent application domain, a particular programing model, and two interesting large scale systems will give us good experience on the applicability of these ideas.

The reason we introduced SESA in this paper was to give the reader at least one concrete model of how an elastic system could be architected. The real purpose of this paper is as a call to the systems community to start focusing on elasticity. Cloud computing makes whole new classes of elastic applications, applications that can exploit thousands of nodes for minutes or even seconds, possible. The nature of cloud computing makes it practical to develop whole new systems, targetting elasticity, and have them be useful for some workload while those system co-exist with legacy systems and applica-

tions. Research in elastic system software is both urgent and can quickly be made relavent to a broad community.

This material is based upon work supported in part by the Department of Energy Office of Science under its agreement number DE-SC0005365 and upon work supported in part by National Science Foundation award #1012798.

References

[1] Amazon. Amazon EC2 Pricing. `http://aws.amazon.com/ec2/pricing/`.

[2] Glenn Ammons, Robert W. Wisniewski, Jonathan Appavoo, Maria Butrico, Dilma Da Silva, David Grove, Kiyokuni Kawachiya, Orran Krieger, Bryan Rosenburg, and Eric Van Hensbergen. Libra. In *Proceedings of the 3rd international conference on Virtual execution environments - VEE '07*, 2007.

[3] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. FAWN: A Fast Array of Wimpy Nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09*, 2009.

[4] Jonathan Appavoo, Volkmar Uhlig, and Amos Waterland. *Project Kittyhawk: building a global-scale computer*. ACM Press, 2008.

[5] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009.

[6] Christian Clmenon, Bodhisattwa Mukherjee, and Karsten Schwan. Distributed shared abstractions (dsa) on multiprocessors. *IEEE Transactions on Software Engineering*, 1993.

[7] Google. Google App Engine. `http://code.google.com/appengine/`.

[8] Philip Homburg, Maarten Van Steen, and Andrew S. Tanenbaum. Distributed shared objects as a communication paradigm. In *In Proc. of the Second Annual ASCI Conference*, 1996.

[9] Orran Krieger, Phil McGachey, and Arkady Kanevsky. Enabling a marketplace of clouds: Vmware's vcloud director. *SIGOPS Oper. Syst. Rev.*, 2010.

[10] Orran Krieger, Mark Mergen, Amos Waterland, Volkmar Uhlig, Marc Auslander, Bryan Rosenburg, Robert W. Wisniewski, Jimi Xenidis, Dilma Da Silva, Michal Ostrowski, Jonathan Appavoo, and Maria Butrico. K42. In *ACM SIGOPS Operating Systems Review*, 2006.

[11] Rose Liu, Kevin Klues, Sarah Bird, Steven Hofmeyr, Krste Asanović, and John Kubiatowicz. Tessellation: space-time partitioning in a manycore client os. In *HotPar*, 2009.

[12] Mesaac Makpangou, Yvon Gourhant, and Jean pierre Le Narzul. Fragmented objects for distributed abstractions. In *Readings in Distributed Computing Systems*, 1992.

[13] Edmund B. Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt. Helios: heterogeneous multiprocessing with satellite kernels. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009.

[14] John Ousterhout, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, Ryan Stutsman, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, and Guru Parulkar. The case for RAMClouds: scalable high-performance storage entirely in DRAM. *ACM SIGOPS Operating Systems Review*, 2010.

[15] P. Patchin, H.A. Lagar-Cavilla, E. de Lara, and M. Brudno. Adding the Easy Button to the Cloud with SnowFlock and MPI. In *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing*, 2009.

[16] Sage. Sage: Open source mathematics software. `http://www.sagemath.org/`.

[17] Matt Welsh, David Culler, and Eric Brewer. SEDA. In *Proceedings of the 18th ACM symposium on Operating systems principles (SOSP '01)*, volume 35, page 230, December 2001.

[18] David Wentzlaff, Charles Gruenwald, Nathan Beckmann, Kevin Modzelewski, Adam Belay, Lamia Youseff, Jason Miller, and Anant Agarwal. An operating system for multicore and clouds. In *Proceedings of the 1st ACM symposium on Cloud computing - SoCC '10*, 2010.

[19] L. Youseff and D. Da Silva. Towards a Unified Ontology of Cloud Computing. In *Grid Computing Environments Workshop 2008*, 2008.