Enabling Automated HPC / Database Deployment via the AppScale Hybrid Cloud Platform

Chris Bunch Chandra Krintz
Computer Science Department
University of California, Santa Barbara
{cgb, ckrintz} @ cs.ucsb.edu

1. ABSTRACT

In this paper, we discuss a prevalent issue facing the HPC community today: the lack of automation in the installation, deployment, and integration of HPC and database software. As a result, scientists today must play a dual role as researchers and as system administrators. The time required for scientists to become proficient with software stacks is significant and has increased with the complexity of modern systems such as cloud-based platforms and infrastructures.

However, cloud computing offers many potential benefits to HPC software developers. It facilitates dynamic acquisition of computing and storage resources and access to scalable services. Moreover, cloud platforms such as AppScale abstract away the underlying system and automate deployment and control of supported software and services. As part of this project, we have extended AppScale with domain specific language support called Neptune that gives developers straightforward control over automatic configuration and deployment of cloud applications. Neptune also extends cloud support beyond web-services to HPC applications, components, and libraries. We discuss AppScale and Neptune, and how they can be extended via more intelligent database usage to provide a better solution for the next-generation of cloud-based HPC and data-intensive applications.

Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Software Engineering - Language Classifications (Extensible Languages); C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks - Distributed Systems (Distributed Applications)

General Terms

Design, Languages, Performance

HPCDB'11, November 18, 2011, Seattle, Washington, USA.

Keywords

Cloud Platform, Service Placement, Domain Specific Language

2. INTRODUCTION

High Performance Computing (HPC) aims to solve problems normally intractable on a single computer via the use of large numbers of machines (e.g., clusters, datacenters, or supercomputers). Over several decades different types of machine architectures, software frameworks, and programming languages have been developed to aid the HPC community in using computing to work on scientific problems. Throughout this period, one problem has remained persistent to HPC users: the high barrier-to-entry imposed by the lack of automation in software packages.

The lack of automation has meant that scientists looking to use HPC resources have had to become system administrators to understand how to use the large numbers of machines typically involved for HPC applications. This unnecessarily takes time away from their research and adds a barrier-to-entry to scientists who want to use HPC resources but lack the advanced technical skills of a system administrator. Further time must be dedicated to sharing data produced by scientific experiments, and the inability to do so trivially stifles the ability of others to verify and reproduce their work.

Social networking software created over the last decade has proven that automated data sharing is possible. Like the users of social networking applications, HPC users need a simple service that they can save the outputs of their programs to and retrieve new inputs from. This service should be able to act as a replicated, distributed, fault-tolerant data storage mechanism that provides easy access for scientists, the community at large, and other HPC programs. Distributed file systems and databases have therefore been natural choices for this job: file systems like NFS and Lustre [11] have been used by the HPC community to varying levels of success, while databases like MySQL and, as of recent, NoSQL datastores like HBase [9] and Cassandra [3] offer easier interfaces and greater data access mechanisms (e.g., structured data formats, query languages) for users of various programming languages.

Some of these software packages can be installed with a single command-line instruction, as their installation process is largely automated, but many are not, requiring scientists who want to use these highly tuned software packages to learn processes like kernel recompilation, which exists well beyond the knowledge base that is comfortable to scientists

at large. Furthermore, for software packages that provide automated installation (which is not the majority of those in use by the HPC community), deploying the software in a distributed setting is often equally difficult. For databases, this can require scientists to learn the optimal parameters for settings such as the block size of the underlying virtual file system (in the case of HBase and Hypertable), the maximum number of clients that the database should support (a common database configuration option), or the desired level of consistency for the data itself. Setting any of these variables (as well as dozens more that each database exposes to users) incorrectly can significantly degrade the performance of applications using it, and a scientist who has not used these technologies before is unlikely to know how to configure these databases for optimal use.

The problem of installation and optimal deployment is exacerbated in the case of newer technologies, such as the NoSQL datastores, which have been around for less time than databases like MySQL. This is not advocating against their use: it is simply the case that less "best practices" are available for newer technologies, and as these software packages mature, experimentation will reveal how to tune them to get them working with HPC codes in an acceptable manner. For example, the Cassandra datastore aims to yield higher performance to programmers at the cost of strong consistency, but to acquire these performance gains, programmers must rewrite their applications to keep in mind that the data they receive might not be completely up-to-date. This different mindset is difficult for seasoned programmers to accommodate, and it is reasonable to assume that it is just as difficult (if not more so) for scientists.

We see a solution to these problems in the cloud computing realm. Here, resources can dynamically (and inexpensively) be acquired and released as needed for computation and storage by users via web interfaces or within their programs themselves. Cloud computing is typically broken down into three layers of abstraction. The lowest level of abstraction, infrastructure-as-a-service (IaaS), provides access to scalable computation and storage resources. Amazon Web Services, through their Elastic Compute Cloud offering, gives users rapid access to large numbers of virtualized compute instances that can be spread out across multiple geographic locations if needed. At a higher level of abstraction exists platform-as-a-service (PaaS), which exposes scalable APIs that developers write programs against. Google App Engine is a prominent force in this field, allowing users to write web applications written in Python, Java, or Go, and upload them to Google to utilize their resources.

It is our belief that the greatest amount of promise lies at the PaaS layer. This middle ground builds on top of the IaaS layer and thus can customize virtual machines it provides with any software package that end users may require, exposing only the APIs required to get the job done. It also has more flexibility than the SaaS layer, which only provides a single software package. It is therefore viable to offer a PaaS that integrates HPC and database software packages, and does so in a way that allows HPC software packages to publish their outputs to distributed, fault-tolerant database. The platform can then make the data available automatically to scientists or other programs. This automation is crucial, and allows the public to utilize the data in their own experiments.

3. CLOUD INFRASTRUCTURES FOR HPC

This system we propose is not tied to anything cloud-specific, but utilizing cloud resources will enable a greater class of scientists to use this system. This is largely a matter of access: most scientists working at research facilities have some access to shared cluster resources, but oftentimes find an inadequate number of resources available for the work they need to run experiments on. Additionally, this work fits in nicely with the "bursty" style of access that the cloud specializes in: scientists we have encountered often only need a few resources at all times to develop their code on, and many resources only when they are ready to run large-scale experiments. This use case also makes better use of cloud resources from a monetary standpoint than statically acquiring a large number of machines for a grid and leaving them idle for most of the time.

Cloud infrastructure providers such as Amazon Web Services [1] have seen some use by scientists to date. However, the consensus among the HPC community appears to be that since infrastructure providers employ virtualization, which necessarily involves a performance degradation depending on the workload involved (e.g., CPU, memory, disk I/O, or network I/O) and virtualization technology employed (e.g., Xen, KVM, VMware), that virtualization is thus an impediment to solving larger problems than a furtherance. This is exacerbated by the opaqueness of the cloud itself: because resources are meant to be interchangable, users often cannot specify that resources are meant to reside in close physical proximity to one another. This is in great contrast to the grid computing model HPC scientists are familiar with, in which they can assume that the machines they are operating with are in the same datacenter and enjoy a low latency with one another (often being connected with high-speed network technologies such as Infiniband).

Cloud infrastructure providers have made some progress towards removing opacity in exchange for greater performance. Amazon Web Services enables users to pay for access to "Cluster Compute Instances", a special type of resource that is guaranteed to be colocated within close range of other instances of the same type (and thus enjoy low network latencies), and boasts more CPU and memory for HPC jobs in exchange for more than an order-of-magnitude price compared to the low-end "Small" instance type. Yet the ultimate customization is offered by private cloud infrastructures like Eucalyptus [12], which facilitate the use of any type of hardware to be used to construct a cloud infrastructure. This use case is a prevalent one to date within scientific projects, as now machines that a group or lab operates can be repurposed to be served and managed automatically without a system administrator. The downside of this approach is that this requires a dedicated cloud system administrator. If the cloud deployment is sufficiently large, the cost may be justifiable, but it may not be feasible for small private cloud deployments.

Another worry that prevents scientists from more actively seeking the use of cloud technologies is that the infrastructure is simply out of their control. If the machines fail, they may be unavailable for several hours, and the vendor may simply not release any information about the downtime until long after it has occurred. These downtimes have been seen across cloud infrastructure providers such as Amazon [13] as well as platform providers like Google App Engine [8] and Heroku. In the case of Heroku, their resources were hosted

in a single datacenter offered by Amazon, so instances where Amazon's infrastructure has failed caused Heroku's services to fail as well, leaving Heroku and its customers with no way to restore service to their users [10]. The effect of these failures to scientists is highly variable: if no large scale experiments are being run, the price of failure is low, but in the rare "bursty" periods when the resources are needed, a failure or datacenter outage can have a significant negative effect upon those who have come to rely upon it.

4. ENTER THE CLOUD PLATFORM

Our solution to the problems with utilizing the cloud for HPC has been a consistent one: we advocate the use of an open platform-as-a-service that can harness multiple cloud infrastructures. This gives us the ability to customize the infrastructure to automatically install HPC software and databases, tune them as needed, and be resilient in the face of failures in a single cloud. We have developed an open cloud platform, known as AppScale, that automatically deploys Google App Engine applications over a variety of different datastores, including HBase, Cassandra, and MySQL. A key feature of AppScale is its automation: when used over a cloud infrastructure, software is installed and configured automatically for use over as many machines as the user can afford. When not used over a cloud infrastructure (e.g., when in use over virtualized instances), automation is still supported: users only need specify the IP addresses where their machines are located, and AppScale acts identically to the cloud scenario.

In its initial implementation [4], AppScale offered automation for web applications, but not HPC applications. We thus expanded upon this with Neptune [2], a domain specific language that allows scientists to automatically deploy certain HPC software packages via AppScale. Users write code in Neptune, a superset of the Ruby [14] programming language, and need only tell it where their code is located, what type of code it is (MPI, MapReduce, and so on), and how many machines are needed. Neptune then instructs AppScale to start all the services needed to run that computation (e.g., spawning that many nodes up and configuring them accordingly) and then runs the user's code. Automation is preserved: the scientist does not need to know where the machines are, nor how to recompile kernels or deal with eventual consistency (to cite previous examples).

Database inclusion is also automatic: for codes that produce output via standard out, the output is automatically saved to any of the nine databases supported by AppScale or Amazon S3. As Google Storage and Eucalyptus' Walrus implementation are API-compatible to Amazon S3, users can also utilize Neptune with these storage backends.

Neptune has also shown itself to be extensible: in addition to serving general purpose HPC software packages such as MPI and UPC, Neptune has also been adapted for use by computational scientists seeking automation for their codes. Specifically, their codes are implementations of the Stochastic Simulation Algorithm [7], which employ Monte Carlo methods to simulate biological processes. To enjoy a reasonable amount of accuracy in their simulations, a large number of them must be run with differing seed values, so Neptune is employed as a coordinator. It acquires as many nodes as the scientist asks for, splits up the work evenly, and merges the final results. Scientists require no system administrator experience to use the system, and a preferred storage system is Amazon S3 due to the large number of tools (such as the Firefox plugin ElasticFox) that seamlessly integrate with it to retrieve their results (here, a series of MATLAB graphs depicting the results). Neptune supports the specialized SSA algorithms provided by the Diffusive Finite State Projection algorithm [6] and the doubly-weighted SSA [5] as well as the general purpose SSA implementation provided by StochKit [15].

5. HPC & DB, MEET HYBRID CLOUD

AppScale and Neptune currently do operate over hybrid cloud deployments. However, there are new directions we can take to better serve the intersection of HPC and databases. We propose two approaches: first, we can utilize hybrid clouds to minimize the downtime caused by a single provider failing (or datacenter within a provider). Hybrid clouds have been explored previously, but only the interoperation between multiple cloud infrastructure providers. We propose utilizing multiple cloud platform providers to greater effect. This allows an application written once to run in multple providers who handle scaling automatically, without user intervention. Cloud platforms like Google App Engine allow for programs to be written in Python, Java, or Go, and then be deployed over Google's hardware. Our proposal is to deploy these applications over Google's platform and the AppScale platform, which can then utilize resources via Amazon or Eucalyptus. Application-level logic present within AppScale can then allow for the two sets of resources to act in unison, and the application (running over either platform) can present a web interface by which new jobs can be started or the results of old jobs viewed.

The second approach harnesses AppScale's automated placement support, by which users can specify which nodes fulfill which roles in the system and have them be automatically configured as needed. We propose to use this support more intelligently in hybrid cloud scenarios: for example, we could place a single database node in each cloud and use AppScale's automated configuration support to ensure that the database instances all communicate with one another without user intervention. We can automatically utilize Cassandra's variable read and write policy setttings to speed up write access at the cost of reads for applications favoring it and utilize it as the underlying database in AppScale. For users who do not know the read/write frequency of their application, we can default to a neutral strategy that favors neither or adaptively switch the strategy, which is possible since we have complete database access.

An example use case of this approach is shown in Figure 1. Here, AppScale is deployed in a hybrid cloud strategy across three clouds: one in Amazon EC2's West Coast availability zone, a second in Amazon EC2's East Coast availability zone, and a third in a local, privately managed Eucalyptus cloud. Each cloud contains one database node and one or more compute nodes to ensure that reads and writes can always be performed via a node in the same cloud. Our example utilizes Cassandra's eventual consistency model, in which only a single node needs to be contacted for reads and writes, to minimize the impact HPC clients face when accessing the database. Database nodes are connected across clouds and update via their internal protocols (e.g., the gossip protocol in the case of Cassandra). In the unlikely event of a failure or outage in a single cloud, nodes still survive in other clouds, and if the underlying framework and com-

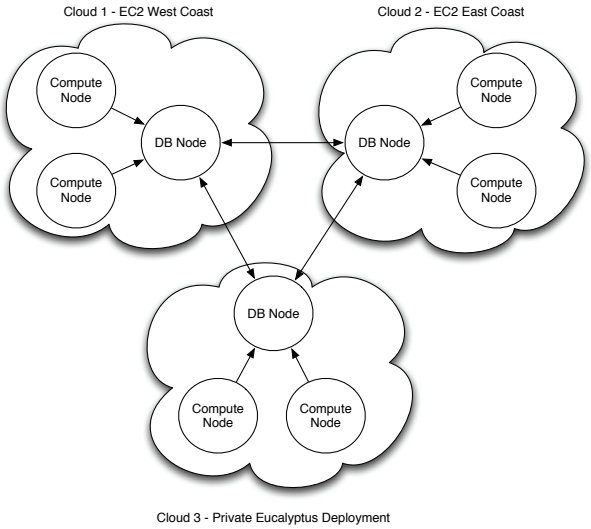

Figure 1: An example of how AppScale's automated placement support can be used to provide better interaction between HPC applications and databases in hybrid cloud deployments. Here, a single Eucalyptus deployment is used when a small number of machines are needed, and two public clouds are used (availability zones in Amazon EC2) when more machines are needed.

putational model is fault-tolerant, as is the case for Hadoop MapReduce, then this failure will not result in the failure of the entire computation.

Two points are critical to make in this example: the automation and potential cost savings. With AppScale and Neptune, this system can be deployed automatically, and does not require the scientist to be an expert user with Amazon EC2, Eucalyptus, Cassandra, and Hadoop MapReduce to be able to run their scientific codes. Furthermore, the scientist can develop and test their code locally against the Eucalyptus deployment, and only when they need more machines, they can be acquired via Amazon EC2.

6. CONCLUSIONS

We see the use of an open cloud platform as a viable means to solve pressing issues facing the HPC community today, especially with respect to automation. Utilizing cloud platforms provides scientists with automated access to HPC resources, including their installation and deployment over independent cloud providers, and alleviates scientists of the burden of knowing how to perform these non-trivially difficult tasks. Furthermore, since a cloud platform could automatically publish and make public the results of scientific jobs, data would be readily accessible via a database that provides data replication and fault-tolerance even if a datacenter fails or becomes inaccessible. Finally, the data stored here is portable: Neptune provides a single interface by which it can store or retrieve data stored in databases it is compatible with, so users can copy their data out of an AppScale-backed deployment running Cassandra and place it in a public cloud-based repository ala Amazon S3 or Google Storage, a private cloud-based repository ala Eucalyptus Walrus, or even another AppScale-backed deployment (with the ten different databases it supports). Additionally, any other databases that the open source community implements support for within AppScale can automatically be utilized by scientists within their codes, without requiring expert database knowledge.

7. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful comments. This work was funded in part by Google, IBM, and NSF grants CNS-CAREER-0546737 and CNS-0905237.

8. REFERENCES

[1] Amazon Web Services. `http://aws.amazon.com/`.

[2] C. Bunch, N. Chohan, C. Krintz, and K. Shams. Neptune: A Domain Specific Language for Deploying HPC Software on Cloud Platforms. In *ACM 2nd Workshop on Scientific Cloud Computing*, 2011.

[3] Cassandra. `http://incubator.apache.org/cassandra/`.

[4] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski. AppScale: Scalable and Open AppEngine Application Development and Deployment. In *ICST International Conference on Cloud Computing*, Oct. 2009.

[5] B. J. Daigle, M. K. Roh, D. T. Gillespie, and L. R. Petzold. Automated estimation of rare event probabilities in biochemical systems. *J. Phys. Chem.*, 2011.

[6] B. Drawert, M. J. Lawson, L. Petzold, and M. Khammash. The diffusive finite state projection algorithm for effficient simulation of the stochastic reaction-diffusion master equation. *J. Phys. Chem.*, 132(7), 2010.

[7] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[8] Java App Engine Outage, July 14, 2011. `http://googleappengine.blogspot.com/2011/07/java-app-engine-outage-july-14-2011.html`.

[9] HBase. `http://hadoop.apache.org/hbase/`.

[10] Heroku Learns from Amazon EC2 Outage. `http://searchcloudcomputing.techtarget.com/news/1378426/Heroku-learns-from-Amazon-EC2-outage`.

[11] Lustre. `http://www.lustre.org/`.

[12] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-source Cloud-computing System. In *IEEE International Symposium on Cluster Computing and the Grid*, 2009. `http://open.eucalyptus.com/documents/ccgrid2009.pdf`.

[13] Amazon Web Services Reports Outage in the U.S. Last Monday. `http://www.pcworld.com/businesscenter/article/237588/amazon_web_services_reports_outage_in_the_us_late_monday.html`.

[14] Ruby language. `http://www.ruby-lang.org`.

[15] K. R. Sanft, S. Wu, M. Roh, J. Fu, R. K. Lim, and L. R. Petzold. StochKit2: Software for Discrete Stochastic Simulation of Biochemical Systems with Events. *Bioinformatics*, 2011.