Scalable Server Provisioning with HOP-SCOTCH

David Daly, Marcio A. Silva, José E. Moreira
IBM T.J. Watson Research Center
Yorktown Heights, NY, 10958, U.S.A.
{dmdaly, masilva, jmoreira}@us.ibm.com

Abstract

The problem of provisioning servers in a cluster infrastructure includes the issues of coordinating access and sharing of physical resources, loading servers with the appropriate software images, supporting storage access to users and applications, and providing basic monitoring and control services for those servers. We have developed a system called HOP-SCOTCH that automates the provisioning process for large clusters. Our solution relies on directory services to implement access control. It uses network boot and managed root disks to control the image of each server. We leverage IBM's Global Storage Architecture to provide storage to users and applications. Finally, interfaces are provided to access the services both programmatically and interactively. We demonstrate the scalable behavior of HOP-SCOTCH by experimenting with a cluster of 40 blade servers. We can provision all 40 servers with a brand new image in under 15 minutes.

1 Introduction

The architecture of choice for many commercial applications is quickly shifting from scale-up (large SMPs) to scale-out (clusters of smaller machines). This change is motivated by a series of factors. First, there is the growing capacity of inexpensive small servers. Second, there is an improvement in the bandwidth and latency of the interconnect between those servers. Finally, there are new classes of applications that simply require too much capacity to be served by anything other than a scale-out architecture.

The chief purpose of the Commercial Scale-Out (CSO) project at IBM Research is to investigate architectures, technologies, and tools that support efficient execution of commercial applications on large clusters of servers. Our previous feasibility studies of a scale-out environment for commercial computing addressed functionality and performance. We investigated the performance implications of using a scale-out environment as opposed to a scale-up environment in [15, 16, 18, 20]. We also investigated the challenges of provisioning the large number of servers needed for a scale-out environment in [7] and image management techniques in [9].

More recently, we have identified a collection of requirements to enable greater adoption of scale-out architectures for commercial applications. First, we must be able to efficiently share and aggregate resources. Second, we must provide strong isolation guarantees between applications and/or customers. Finally, we must ensure high reliability and availability to the applications.

This paper describes our work on enabling the efficient sharing and aggregation of resources. Specifically, it focuses on the efficient provisioning and repurposing of hardware, as described in more detail in the next section. We had previously addressed this topic in [7]. In that paper, we described techniques to completely automate the provisioning of a blade server in a virtualized environment. As blades were inserted into the system, they were discovered, updated, and configured to boot a shared, read-only filesystem over a storage area network (SAN). During boot time, the system also mounted a shared, high performance network file system (GPFS) for application use and the storage of state. That work has been released under the Common Public License (CPL) with the *HOP-SCOTCH*[1] name and is the basis for continuing work.

We have now extended that work by tackling several aspects of the provisioning life cycle. We address the following issues. First, we developed an approach to handle ownership and access of physical resources. Second, we provided storage solutions to users and applications. Third, we addressed the problem of loading the proper software stack on a server. Finally, we provided interfaces to administrators and users to control the physical resources.

Our solution relies on directory services to implement access control to owners and users of physical resources. We leverage an existing IBM solution (Global Storage Architecture – GSA) to provide storage to users and appli-

[1] HOP-SCOTCH is available for download on sourceforge.net, at https://sourceforge.net/projects/hop-scotch.

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

cations running on our system. We use network boot and root disk management to deploy the software stack in the servers. New interfaces are provided to access our services both in a programmatic and interactive way.

The rest of this paper is organized as follows. Section 2 defines more precisely the challenges in managing the physical resources in a scale-out environment. Section 3 describes the approach we took in our solution to those challenges. Section 4 reports the results from experiments we performed in an actual system. Section 5 discuses related work. Finally, Section 6 presents our conclusions and future work.

2 Challenges

A commercial scale-out infrastructure is used by organizations to run applications that support their business. It is common to have a single physical infrastructure shared by several applications and even competing organizations. This sharing nature of resources leads to many challenges that are characteristic to a commercial scale-out system and not very common in scientific computing:

- Resource ownership: the resources of the cluster have to be partitioned into sub-sets, owned by specific organizations, with strong isolation, minimal administrative intervention, and virtually no interruption.
- Non-homogeneous resources: different applications have specific resource requirements and these resources must be available to the applications in an automated way.
- Time-varying load: commercial applications have their resource demands tightly coupled to external factors that change dynamically (e.g., web-based shopping patterns change during the day and from season to season).
- Independence from resource location: the application has to be capable of operating virtually without interruption even if migrated among different components in a given sub-set of the cluster, among sub-sets, among clusters, and even among data centers.

The challenge faced by the CSO project can be clearly stated: the need to share, provision, repurpose, monitor, and control large amounts of computing resources in order to support a set of heterogeneous commercial applications, each one with inherently conflicting and highly dynamic requirements. First, we need to be able to allocate physical resources to specific groups that will use those resources for their applications. Second, we need a mechanism to load those physical resources with an appropriate software stack to perform the application. It must be possible to quickly change that software stack, thus changing the function associated with the resource. Finally, we need to be able to monitor the behavior of those resources and perform basic control operations such as boot, shutdown and grant/revoke access.

We cannot expect to create a single tool that will perform all the functions required to manage a commercial scale-out infrastructure. Rather, the approach must be to create layered and extensible tools that can operate and compose with other tools, sometimes developed by other vendors. The tools must be accessible both through human-friendly interfaces (e.g., web pages) as well as programmatically (e.g., they can be scripted). These guidelines were followed as we developed the HOP-SCOTCH management library for the challenges described above.

3 Architecture

The CSO cluster is a testbed for scale-out commercial computing and is managed by the HOP-SCOTCH library. The cluster architecture consists of various resource pools. The resource pools are the computing, communication, storage, and provisioning pools.

The computing pool uses general-purpose nodes (i.e., blade servers) as elements to provide processor cycles and volatile memory for application execution. The communication pool uses specific purpose nodes (e.g., switches) and general purpose nodes (e.g., a blade server acting as a firewall) to interconnect elements of an application. The storage pool also uses specific purpose nodes (e.g., RAID disk arrays) and general-purpose nodes (e.g., a blade server acting as a NFS server) to allow the computing pool to access the data used by applications. Finally, the provisioning pool is formed by servers running the HOP-SCOTCH management library (currently a single management server), along with all other required services, including remote boot (DHCP, PXE, TFTP), authentication/authorization (LDAP), root disk image (NFS) and access interfaces (SOAP, CLI). A consistent effort was made to centralize the cluster state information in the provisioning pool. By keeping the remaining pools as stateless as possible, the system can quickly respond to resource demand changes by repurposing the resources.

Figure 1 depicts the architecture of the CSO cluster. The physical resources are displayed with their associated resource pool. Blade servers can be freely repurposed among pools, in order to cope with resource demand surges or idle capacity. All resources are exposed to applications through virtual interfaces. The communication and storage components are virtualized by the use of virtual network-capable switches and Fiber Channel (FC) RAID storage controllers. Sets of blade servers are packaged inside chassis. Each blade server is then virtualized with the Xen [5] virtual machine monitor, in order to allow applications inside virtual machines to be deployed on it.

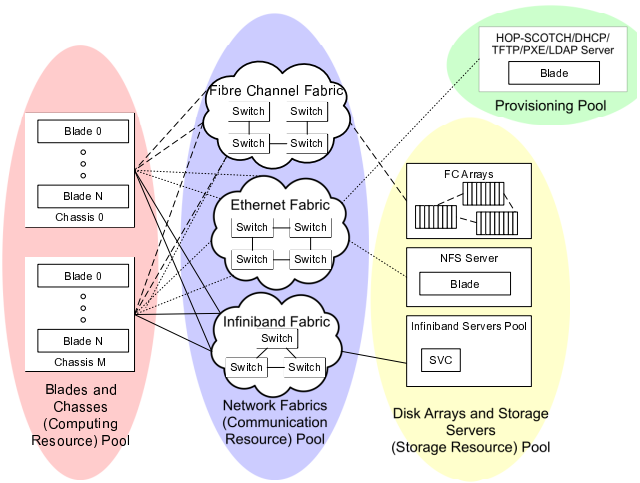

Figure 1. The CSO cluster resource pools.

By virtualizing the resources, it is our intent to allow an application manager to focus solely on the application architecture and topology. This enables a new level of virtualization, in the form of application packages or templates and frees the application manager from the job of configuring physical resources.

3.1 Storage features

The main application and user storage solution we adopted for the CSO cluster was an IBM internal infrastructure called Global Storage Architecture (GSA). IBM employees can use GSA to create personal and project directories that reside in managed storage. Although this is an internal IBM solution, it is representative of managed storage solutions that coorporations can deploy for their own use.

GSA provides two key features, networked storage and user authentication. The GSA client is installed on the base system image used to provision our servers. On a system with the client installed, a user can log into any server using their GSA credentials. The user's home directory on the system is automounted over the network, as are project directories. The user or application is always able to access their own data and access the same shared data. The GSA authentication database is also used in the LDAP authentication schemes described in Section 3.3

We also found it useful to have a second form of network storage available without the GSA authentication. The base system image also mounts an NFS directory from the management server. The NFS based solution is of particular use as scratchpad storage between the servers and as a small store for application data that is preserved between migrations and reprovisionings. The NFS based system could also be used more extensively in a installation without an enterprise storage solution such as GSA.

Other storage support has been added or expanded in the tool, but is not discussed in detail in this paper. This functionality includes automated support for SAN storage and GPFS in a cluster. The tool can automatically map and zone servers so certain luns are accessible. Additionally, the system can automatically add or remove GPFS support to a server.

3.2 Non-local root disks

We previously implemented a shared root filesystem approach [7]. We found that the heterogenous nature of commercial applications led to small customizations needed for individual groups of servers, different versions of the same base root file system, as well as frequent updates to it. Accommodating this heterogeneity in operation became onerous in certain situations. We decided to pursue an improved non-local root disk solution. Specifically, the solution had to be storage efficient, capable of being performed by any server in the CSO cluster, require as little state information and configuration as possible, and have the ability to be shared among a large group of servers.

The most mature storage candidate for our requirements is NFS. Using an NFS-exported directory as the root filesystem for a server is well documented and tested [1, 8]. NFS provides a file-based access mechanism both to the server using it as its root filesystem, and to the administrator on the filesystem server, enabling quick modification of a root file system without having to mount it. Additionally, it makes strong usage of caching, thus reducing the bandwidth requirements for clients.

The NFS server was implemented as a regular server that accesses a set of root filesystems in a storage controller via Fiber Channel (FC). We chose to use a standard server for the NFS server instead of using a specific-purpose NAS device mainly because it allows any blade to perform this role. We configure (using HOP-SCOTCH) a blade to boot from a root disk on the FC storage and access the storage volume containing all root file systems. Since this storage volume can be shared, a configuration with a high availability (HA) cluster can be used in order to eliminate a single point of failure.

Even with an NFS solution there is still the issue of choosing a native file system format that can then be NFS exported. While NFS is a robust root file system solution, it has no intrinsic capability for storage consolidation. Since the number of images grows both with the number of servers and with the number of types, a scalability issue arises.

We decided to use the ZFS [2] file system, part of the OpenSolaris operating system, to address the storage effi-

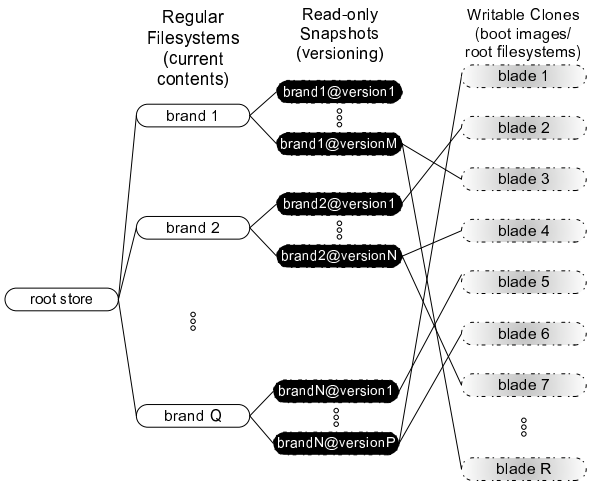

Figure 2. Structure of NFS root file systems.

ciency requirement. ZFS implements "snapshots", a read-only consistent virtual image of a file system at a given point in time. ZFS then supports the creation of "writable clones" from a snapshot. A writable clone can be modified and only requires storage space for the data that is modified. Both snapshots and writable clones can be created very quickly (i.e, a few seconds) and require minimal storage space upon their creation (i.e. hundreds of kilobytes), thanks to the use of copy-on-write [4] technology.

We developed a naming scheme (Figure 2) based on the need to maintain different types of root file systems for different purposes. A root type created for a specific purpose is labelled with a brand. Different brands can be created for various administrative, technical, and operational needs. The use of different brands allows a subset of resources in the CSO cluster to be effectively compartmentalized for an organization. A root brand can experience several revisions throughout its lifetime. Since such revisions are required for an assortment of purposes, we make use of snapshots to implement a versioning mechanism.

HOP-SCOTCH maintains a brand's root directory as a "golden image", and is capable of creating on-demand snapshots of this golden image. An incremented version number is appended to each snapshot. In this way, changes made to the golden image are captured, stored, and made available in an efficient way, and the changes can be rolled back if required. Writeable clones are made from the snapshots to create the root file systems presented to the servers. The capability to quickly create a clone from a snapshot from an arbitrary brand gives the CSO cluster the ability to quickly reprovision and repurpose resources. Because the reprovisioning process is so efficient, the root file systems can even be destroyed and recreated at each reboot. This assures a consistent state after a power cycle.

After a base golden image is created, the provisioning process comprises three steps: cloning, sharing, and rebooting. The cloning phase is time constant and requires the specification of a server name, a brand, and its version for the creation of a writable clone. The sharing phase consists of configuring the NFS server to export the writable clone and configuring the target blade to boot from the network. When those two steps are complete, the blade can be booted immediately.

3.3 LDAP Authentication

A major focus of the work this year was the addition of user authentication and the concept of ownership of servers. In the past we had used ad-hoc techniques to allocate the ownership of servers within our own cluster, leading to inefficient use of the servers and contention for them.

There are several requirements for an authentication system. First and foremost it has to be able to allow for user authentication and for users to atomically checkout or release servers. Additionally, another repository of user accounts should not be created if it can be avoided. Finally, the authentication system should honor the general system design principals used in the rest of HOP-SCOTCH, namely the system has to be extensible, user accessible, and programmatically accessible in as open a way as possible.

The server ownership data requires a database of some form to store the current ownership data. We decided to use an LDAP [19, 13] directory server for the ownership information database. The LDAP directory server comes with several advantages. First, it has explicit support for atomic operation on records. Second, it is inherently a network service with clean access protocols, enabling remote access and programmatic use. Third, in many cases there already is an existing LDAP directory for user accounts that can be extended for the server ownership. We used the OpenLDAP server for our system, but any LDAP directory should have sufficed.

Three essential pieces of information need to be stored in the LDAP directory: (i) the server name, (ii) the server owner (decides who can operate the server) , and (iii) members who are allowed to operate the server. We use the *GroupOfNames* schema from the OpenLDAP core schema to store this information. The server name is the common name (cn), the server owner is the *owner*, and the members are the *members*. Other schemas representing collections of users would have also worked.

In addition to the servers, we also need to specify the users in the system. However, many organizations have existing LDAP directories for user information. To address the problems of existing user accounts, we exploited OpenLDAP's ability to proxy other LDAP servers. The proxy database passes along certain LDAP queries to an-

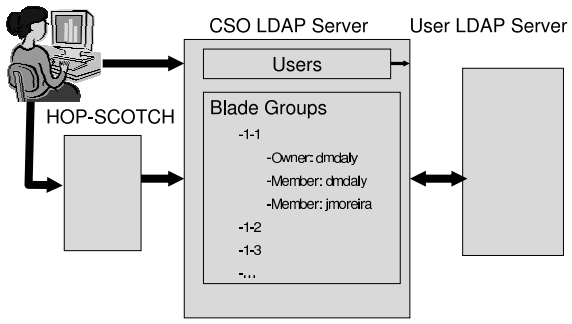

Figure 3. HOP-SCOTCH LDAP architecture.

other LDAP server. We configured a proxy database to our enterprise LDAP server. The architecture is demonstrated in Figure 3, including the possibility of a user accessing the database through the HOP-SCOTCH libraries, or directly.

Using the proxy database back-end, a user is authenticated on our LDAP server by passing along the authentication request to the enterprise LDAP server, allowing us to fully leverage the existing infrastructure.

A third aspect of the server database is enforcing useful security semantics on the server data. Given that a server has been checked out, we want only the owner of that server to update the server's entry in the database, such that another user cannot steal the server or give oneself access to the server. We also want to make sure that a checkout of a server is atomic, and a second checkout request that arrives during a previous request fails. Additionally, we want to ensure that no user can put the entries into a bad state, such that others cannot use the server.

We were able to meet all those requirements through specifying appropriate permissions on the entries. OpenLDAP has an ordered permission scheme in which a number of policies can be specified, and the first matching policy is used. Additionally, a permission scheme can be matched against a set of entries based on location in the database, as well as upon values in the entry itself.

The following summarizes the permission scheme that we implemented. Note that the first matching policy is used to determine access, so the rules need to be considered in the order listed: (1) Unauthenticated users have no access. (2) For a server with an owner, the owner can write or clear their own name from the owner field and all users can read the owner field. (3) For a server without an owner, all users can write their own names in the owner field and all users can read the owner field. (4) For all servers, the owner can write to any field and all users can read any field.

The above policy ensures the properties we desired. For instance, any authenticated user can checkout a server by writing their own name in the owner field. However, if someone already owns the server, the second rule applies, and the user will be denied write access to the owner field.

Similarly, the permission scheme prevents a user from transferring a server to a non-existent user. The second rule explicitly limits what the owner of a server can write in the owner field. The owner can only clear the entry and cannot write someone else's name. The owner is allowed to write to any other field according to the last rule, but the second rule takes precedence on access to the owner field.

The system as described above allows for the normal checkout and release of servers in the system. However, it does not address certain administrative policies that are likely desired. It allows any user to check out any number of free servers and hold on to those servers as long as they please. That level of freedom is clearly not desired in any shared computing infrastructure. As such, we include an administrative account. The administrative account has complete read and write permission on the server entries. Therefore, the administrator may release servers forcefully when needed, and transfer those servers to other users as needed. One could also use the override feature to implement various administrative policy, such as limiting the duration a server checkout.

The design of the LDAP based user authentication and server checkout implements the entire checkout policy within the LDAP directory. This enables at least two distinct use cases: checkout through HOP-SCOTCH tool or checkout through direct interaction with the LDAP directory server. The primary use is for the HOP-SCOTCH libraries to use the LDAP directory information. We've implemented additional command line switches and checks to check out servers, query server state, and to verify that a user has permission to use the server before processing commands.

However, the way the authentication functionality was implemented enables the direct access to the LDAP directory server itself. A user can checkout a blade through the direct interaction with the LDAP server. Additionally, the LDAP interface can be programmatically controlled. We use this ability in the following section when building web page and web services interfaces to HOP-SCOTCH.

3.4 Web Services and Web Interface

We extended HOP-SCOTCH to have more interfaces through the use of web services and a web interface. HOP-SCOTCH has always been designed with a goal of extensibility: its constituent libraries have been written in a layered manner, exposing functionality at many levels. A user can call the HOP-SCOTCH functionality remotely, and is able to build extensions on any computing platform. Our approach is to expose HOP-SCOTCH as a web-service using SOAP [11] (Simple Object Access Protocol) and WSDL [6] (Web Services Description Language). Additionally, to increase the usability of the tool, we developed a web inter-

face to HOP-SCOTCH leveraging the SOAP interface and the LDAP interface.

SOAP is an Internet messaging protocol standard built upon XML, and enjoys wide support, including support in most major programming languages. We have used SOAP messages in a simple request and response pattern. A remote client can submit a SOAP message across the network specifying an operation, options, and targets to the HOP-SCOTCH library. The management server performs the operation and returns the requested information to the client.

WSDL is a companion language to SOAP. WSDL is used to specify the operations supported by a service, and the specification of the SOAP messages involved in performing the operation. Given the WSDL document describing the HOP-SCOTCH service, any desired architecture can be used to generate and parse the SOAP messages needed to remotely access the HOP-SCOTCH functionality.

Python supports SOAP and WSDL through a few tools, including the Zolera Soap Interface (ZSI). We used the ZSI tools to implement the SOAP interface to HOP-SCOTCH. Unfortunately, ZSI does not have support for generating WSDL documents for existing functions[2], but it does support making a stub code from a given WSDL document. Therefore we manually created the WSDL document describing the available HOP-SCOTCH functions. The WSDL document specifying the exposed HOP-SCOTCH functionality is available on the project site.

The addition of LDAP authentication and SOAP interface provides a great deal of flexibility for extending HOP-SCOTCH. We have added a web interface leveraging the SOAP and LDAP interfaces in addition to the existing command-line interface. The addition of the web interface has two-fold value: it greatly simplifies the use of the HOP-SCOTCH system for certain tasks, and it demonstrates the flexibility available in extending the HOP-SCOTCH functionality in other languages.

We wrote a number of web pages using PHP. The PHP code handles authentication with the system, and directly accesses the LDAP and SOAP interfaces to HOP-SCOTCH. We list two examples of pages in the web interface. One page enumerates all available servers in the system, listing details about each server, and allowing the user to check out a subset of the servers. Figure 4 shows another page that presents all the servers the given user owns, along with buttons to control the server (such as powering off or rebooting the blade). We found that the web pages proved to be much more convenient for server checkout procedures than the command-line and generally useful.

Initially, we experienced a problem of pages taking too long to load because of the SOAP commands. We were

[2]SOAP messages are strongly typed, while Python is not. Since Python is not strongly typed it is hard to determine the types needed in the SOAP messages.

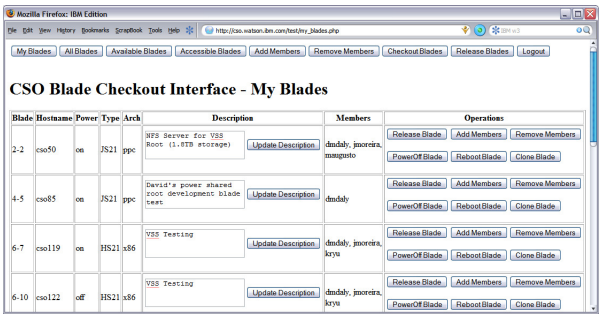

Figure 4. HOP-SCOTCH web interface.

able to rewrite the affected pages with the addition of JavaScript to make the SOAP commands execute asynchronously, making the web interface much more responsive.

4 Experimental results

As a proof-of-concept, we report results from using HOP-SCOTCH in a system that we developed for a potential customer. The system consists of a rack with 3 BladeCenter-H chassis, each with 14 blades. The blades are dual-processor/dual-core (total of four cores per blade) with 16 GiB of memory. Some of the blades have internal disk, but for the purpose of our experiments we treat all the blades as diskless. That is, we never use their internal disks.

In addition to the blade chassis, our system contains a "master node" from which we perform the control operations and an NFS server node that manages the root file systems for the blades in a ZFS file system. Storage for that NFS server is in the form of a Fiber Channel-attached RAID box. The ZFS file system contains three brands of golden images, which can be used for provisioning the blades. Each brand has on average 5 versions. For the experiments, we use a golden image with approximately 140 thousand files and total size of 9.7 GB, for an average file size of 70 kB.

Using the above configuration, we perform provisioning (cloning and booting) of different sets of blades. We use sets of 1, 5, 10, 20 and 40 blades. We monitor both network traffic and the NFS server activity during clone and boot. The cloning operation is performed serially. That is, we finish cloning one blade before proceeding to the next. The boot operation is performed in parallel. We initiate the boot operation in a blade, wait 5 seconds and then move to the next blade without waiting for the boot to complete. The 5-second interval between blades helps stagger the boot process.

Figure 5 shows clone and boot times for different numbers of blades. We observe that clone time increases linearly with the number of blades. This is expected since

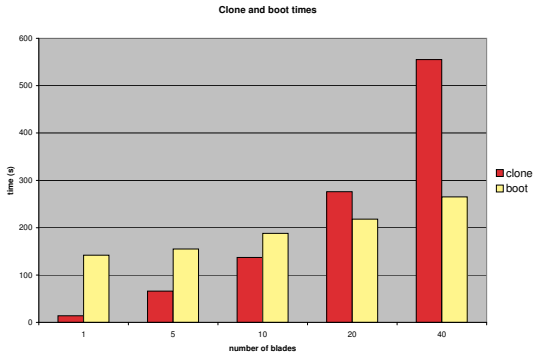

Figure 5. Clone and boot times.

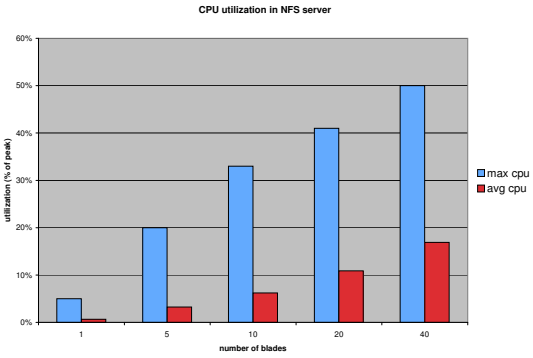

Figure 6. NFS server utilization during boot.

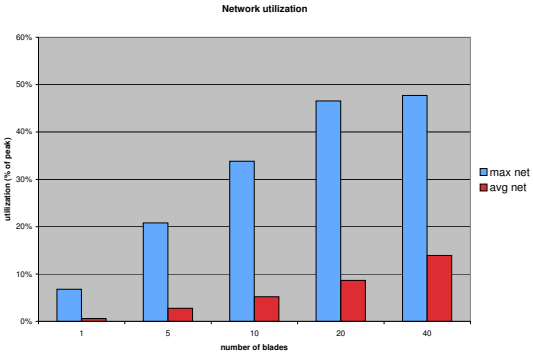

Figure 7. Network utilization during boot.

we perform those operations sequentially. The time is approximately 14 seconds per blade, independent of image size (since ZFS does copy-on-write for the clones). We attempted to perform multiple cloning operations in parallel but we ran into stability problems with our servers. Nevertheless, our results show that we can prepare the images for all the blades in our configuration in approximately 10 minutes. The boot time is sub-linear with the number of blades. In fact, we can boot one blade in approximately 2 minutes and 40 blades in only 4 minutes. The cloning time starts to be the dominant time for provisioning with about 20 blades. We note that we can go from a completely idle system to all blades provisioned and running in less than 15 minutes.

We also want to understand in more detail the behavior of the NFS server during boot. We know that the server is not very busy during cloning (CPU utilization stays between 1-2%) and it is the boot operation that has the potential to saturate the server. Figure 6 plots both the maximum and the average CPU utilization of the NFS server during the boot process, for different numbers of blades. As expected, both the maximum and average CPU utilization of the server increase with the number of blades. We note that the maximum reaches only 50% even with all the blades booting in parallel. Furthermore, the average utilization is less than 20% with 40 blades. This demonstrates that the NFS server still has plenty of capacity to serve more blades in parallel.

Another important resource during the boot is the Ethernet network connecting the blades to the NFS server. Figure 7 plots both the maximum and average network utilization during the boot process, for different numbers of blades. We observe that the network seems to be saturating at about 50% utilization. This is understandable, since the average network packet size during boot is relatively small (under 800 bytes). Although we may not be able to drive the maximum network utilization beyond what we are already achieving, the average network utilization is relatively modest, staying under 15% even with 40 blades. Again, this indicates that we can support even more blades with our current NFS server infrastructure.

We demonstrate the storage efficiency of ZFS by reporting the disk usage in our approach. If we were to store all versions of the three brands we use explicitly, that would require 286 GB. With ZFS we use only 52 GB. Furthermore, preparing 40 clones of each golden image brand (one clone per blade that we want to deploy) we would need approximately 400 GB of storage per brand per version. With the ZFS copy-on-write approach, the total storage we use for all versions of a given brand is only 25 GB, resulting in a substantial reduction in storage needs.

5 Related work

The work reported in this paper builds upon and is related to a nuber of other projects. A number of tools exist to support the low level provisioning of servers, including IBM's Remote Deployment Manager, PXE boot, and the Red Hat network install process (Anaconda). All of these tools can be used to install an operating system on a server. Additionally, there are a number of packages for installing HPC clusters, including [12, 17].

Additionaly, there are many papers discussing the challenges and issues for a commercial scale out infrastructure (cluster), as well as its many incarnations: "utility data center" [14], "computing utility" [3], "virtual data center" [10].

One common key aspect on all these previous compilations and descriptions are the heterogeneity required for this kind of cluster, both in time (i.e., changing during the cluster's cycle of operation) and space (i.e., in specific portions of the cluster).

6 Conclusions

In this paper we have presented our recent work in extending the HOP-SCOTCH tool with a scalable server provisioning tool. We addressed the problem of sharing physical resources with an LDAP-based solution for controlling ownership and access privileges. We support the rapid installation of software images in servers by means of a ZFS/NFS infrastructure for the root file system of those servers. We used GSA to provide users and applications in those servers with persistent storage. Finally, we developed new interfaces for users to control and monitor the resources by leveraging web services standards.

Our experimental results strongly support the evidence that we can manage an entire rack of blades, which can accomdate at most 56 blades, with a single ZFS/NFS server. This leads to a scalable approach with each rack having its own root file system server and the collection of images that can be used to provision the blades.

There are several areas of future work for us. First, we want to explore more parallelization in our tools. We demonstrated good parallelism in the booting process of blades, but the cloning process still has stability issues that force us to operate serially. We also want to investigate other approaches to root file systems, including the possibility of using only a RAM-based root file system for at least some cases. Finally, and most importantly, we want to deploy HOP-SCOTCH in environments with more users and more machines, to demonstrate its effectiveness in real production scenarios.

References

[1] NFSroot HOWTO. *http://www.onesis.org/NFSroot-HOWTO.php*, 2005.

[2] Solaris ZFS Administration Guide. *http://docs.sun.com/app/docs/doc/819-5461*, 2007.

[3] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano – SLA based management of a computing utility. *2001 IEEE/IFIP International Symposium on Integrated Network Management*, pages 855–868, 2001.

[4] A. Azagury, M. Factor, W. Micka, and J. Satran. Point-in-time copy: Yesterday, today and tomorrow. *Proceedings of the Tenth Goddard Conference on Mass Storage Systems and Technologies*, April 2002.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM Symposium on Operating Systems Principles (SOSP), 2003*, October 2003.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1*. 2001.

[7] D. Daly, J. Choi, J. Moreira, and A. Waterland. Base operating system provisioning and bringup for a commercial supercomputer. In *Third International Workshop on System Management Techniques, Processes, and Services (SMTPS), IPDPS 2007. Long Beach, CA, March 30th*, 2007.

[8] H. de Goede. Root over nfs clients & server Howto. *http://www.faqs.org/docs/Linux-HOWTO/Diskless-root-NFS-HOWTO.html*, 1999.

[9] O. Goldshmidt, B. Rochwerger, A. Glikson, I. Shapira, and T. Domany. Encompass: Managing functionality. In *Third International Workshop on System Management Techniques, Processes, and Services (SMTPS), IPDPS 2007. Long Beach, CA, March 30th*, 2007.

[10] S. Graupner, V. Kotov, and H. Trinks. Resource-sharing and service deployment in virtual data centers. *22nd International Conference on Distributed Computing Systems Workshops*, pages 666–671, 2002.

[11] M. Gudgin, M. Hadley, N. Mendelsohn, J. J. Moreau, and H. F. Nielsen. Soap version 1.2. *W3C Working Draft*, 2002.

[12] E. Hendriks and R. Minnich. How to build a fast and reliable 1024 node cluster with only one disk. *The Journal of Supercomputing*, pages 171–181, May 2006.

[13] T. A. Howes. The lightweight directory access protocol: X.500 lite. Technical Report 95-8, Center for Information Technology Integration, University of Michigan, July 1995.

[14] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler. SoftUDC: a software-based data center for utility computing. *Computer*, 37(11):38–46, 2004.

[15] M. Michael, J. E. Moreira, D. Shiloach, and R. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Third International Workshop on System Management Techniques, Processes, and Services (SMTPS), IPDPS 2007. Long Beach, CA, March 30th*, 2007.

[16] J. H. Tseng, H. Yu, S. Nagar, N. Dubey, H. Franke, P. Pattnaik, H. Inoue, and T. Nakatani. Performance studies of commercial workloads on a multi-core system. In *IEEE International Symposium on Workload Characterization (IISWC 2007)*, pages 57–65, 2007.

[17] A. Wachsmann. A general purpose high performance Linux installation infrastructure. Technical report, SLAC, November 2002. SLAC-PUB-9193.

[18] R. W. Wisniewski, R. Azimi, M. Desnoyers, M. M. Michael, J. Moreira, D. Shiloach, and L. Soares. Experiences understanding performance in a commercial scale-out environment. In *Euro-Par 2007. Lecture Notes in Computer Science, vol 4641*, pages 139–149, 2007.

[19] W. Yeong, T. Howes, and S. Kille. RFC1777: Lightweight Directory Access Protocol. *Internet RFCs*, 1995.

[20] H. Yu, J. E. Moreira, P. Dube, I. Chung, and L. Zhang. Performance studies of a WebSphere application, Trade, in scale-out and scale-up environments. In *Third International Workshop on System Management Techniques, Processes, and Services (SMTPS), IPDPS 2007. Long Beach, CA, March 30th*, 2007.