Dual-layer hardware and software management in cluster systems

Sergii Stirenko[1], Oleksandr Zinenko[1], Dmitri Gribenko[1]

[1]National technical university of Ukraine "KPI", Kyiv, Ukraine

[sergii.stirenko, oleksandr.zinenko, dmytro.hrybenko]@hpcc.kpi.ua

Abstract. In this paper, we describe the software architecture of a cluster system, which allows to reassign any particular system task to another hardware without relying on virtualization. We also show the support for services virtualization along with non-virtualized computational nodes.

Keywords

Cluster management, cluster software, virtualization, network booting, software versioning.

Introduction

Extensive utilization of parallel computing, nowadays, is the simplest way to gain in performance. Traditional high-performance clusters comprise multiple homogeneous or at least similar in configuration computers, which are connected in a specialized low-latency network. In order to seamlessly operate as a single machine from the end-user point of view, such clusters use specific software, which allows user to interact with the system on the whole. Nevertheless, each computer has its own operating system and should be, in worst case, configured separately. So while the user sees but a single interface, the cluster owner should deal with a bunch of computers running services along with the large number of computational nodes. With the constant growth of cluster sizes to reach exaflops performance, controlling every single computer becomes infeasible. Therefore, one should propose a unified consistent interface for the cluster administration and management.

Network boot was proposed to avoid repeated configuration of multiple computers and to allow usage of diskless computational nodes, which is preferable for a number of reasons including power consumption. While network boot itself is a clever idea, it makes bandwidth the bottleneck when booting even a middle-sized cluster. Furthermore, network boot capabilities should be extended with operating system configurability in order to deal with emerging heterogeneity of computer systems as modern high-performance computers extensively exploit different kinds of computation accelerators. In this case, one wants to preserve consistency as much as possible between different kinds of nodes, provide similar software and configurations to the user.

Several solutions were proposed by software vendors to address these problems, but most of them are tailored for commercial use, namely to allow accounting on every parameter, which is not the case for an academic usage. In this paper, we propose a software architecture of a cluster system, simplifying centralized cluster management and using exclusively freely available software.

1 Let nodes do computations

Virtualization is often seen as a modern solution of the resource management problem and is widely used to provide computational resources to the clients. Initially virtualization was used to share resources of a single computer between multiple users and to provide an isolated environment as a security feature. Then it evolved to become the main part of cloud computing systems, providing users with virtually unlimited resources on demand.

The benefits of virtualization come at a price of performance overhead required to support the virtualization itself. Depending on the virtualization type and type of activities performed (processor-hungry computations, disk I/O, network I/O), this overhead can reach the values as big as 40%. In HPC most system run Linux operating system, therefore the choice can be biased to the operating system virtualization, which presumably has the lowest overhead thanks to using the single kernel. Nevertheless, this still has a performance overhead of around 5%, quite a significant one in light of petaflops performance.

Table 1. MPI performance in different settings

Value	TCP over Ethernet	TCP over Infiniband	Native Infiniband
Latency, us	220	108	4.6
Real throughput, Gb/s	0.7	1.1	6.2
Message rate, msg/s	$0.4 \cdot 10^6$	$7.3 \cdot 10^6$	$19.1 \cdot 10^6$

Operating system virtualization can be still used even with a decrease in performance, but it has another major drawback: as of now, no virtualization engine of this kinds allows to share PCI devices between virtual machines. As the communication is a bottleneck in the high-performance clusters, most of them use specialized low-latency network, for example Infiniband with PCI cards in every node, to compensate that. Virtualization engines allow either attaching the PCI device to a particular machine or use it as a shared TCP network. The first option heavily affects task granularity as the nodes can no longer be shared between different users willing to perform computation. In worst case, only one processor in each node will be used while every other will be idling, which significantly decreases overall system performance. Furthermore, in light of emerging vast heterogeneity, different compute-capable devices of the same node can be used simultaneously on the same node. Considering this option, one also should take into account that many users require clusters to run multiple serial tasks at once. The latter option hinders low-level network optimization rendering specialized network interfaces useless. Thus, even operating system virtualization should be avoided to reach maximum performance unless special networking optimizations are implemented.

The could computing model, widely developed in the last years, is considered perpendicular to the high-performance computing paradigm: although one can obtain large computing resources in the cloud, no control is possible over the physical location of these resources, which in several cases results in big latencies and low bandwidths between them. This is a major drawback, which prevents usage of cloud resources to eventually extend computational clusters. On the other hand, deploying a private IaaS cloud on a cluster might be considered given the aforementioned virtualization problems solved and cluster topology is taken into account. As for today's technology, application of the most cloud computing features, like elasticity and reliability (based on virtual machines migration), remain open questions for HPC use, as well as device and location independence as a cloud feature should be discarded in this case.

2 Dual-layer startup

Starting up or restarting entirely a large-scale computer designed from multiple nodes is quite a time-consuming task, which can become even longer if there exist several nodes with special function. Both from the user and from the management perspective there is no difference which physical node will perform which task given their hardware is identical. The latter gives the opportunity to abstract away the hardware running particular task.

The aforementioned abstraction is achieved without virtualization by using a two-level network boot schema. As a precondition, a network boot server, containing TFTP, HTTP and iSCSI should be up and running before booting a cluster. First, when a node starts up, it uses its PXE capabilities to obtain a small iPXE image over TFTP protocol from the network boot server. This image is able to perform more complicated tasks, like inspecting network connection parameters obtained via DHCP protocol during initial communication with the network boot server, sending HTTP requests and performing specially prepared script. Second, this image sends an HTTP request containing hardware identifier to the netboot server in order to obtain the boot sequence to initiate. This boot sequence contains a path to iSCSI image (target in iSCSI notation, hereinafter we use image for the sake of consistency) of the system to boot. On the second stage, network boot server is able to identify which physical nodes want to boot and returning a corresponding script. The server can run any HTTP-enabled software allowing to design arbitrarily complex dependencies, for example provide a web-interface to distribute images between nodes. In our implementation, we use a hierarchy of symbolic links support booting the latest image multiple times. To do so, an iSCSI image should be created as read-only. While to allow proper execution of an operating system, /var and /tmp directories use tmpfs while running, which stores changes to their files in

the main memory and discards them when shut down as the read-only image is not copied back to the server. This allows to simply reboot the node to return to the initial state in case of any inappropriate use.

The cluster operation is supported by multiple services, namely by the scheduler and user access management system. This services can be installed to another image and booted once on any hardware following the choice on the second network boot stage. These images are read-write and should be booted only once. To provide better security guarantees via isolation, the services can be put in a separate virtual machines, in which case the bootable image becomes a virtualization-enabled system with predefined set of virtual machines to run as long as other nodes can address these services, which is easily solved by using DNS names instead of fixed IP addresses. Our implementation uses SLURM scheduler and LDAP-based user management, which are executed inside different OpenVZ containers on a single node. On the other hand, we use separate server with virtual machines hosting GRID services.

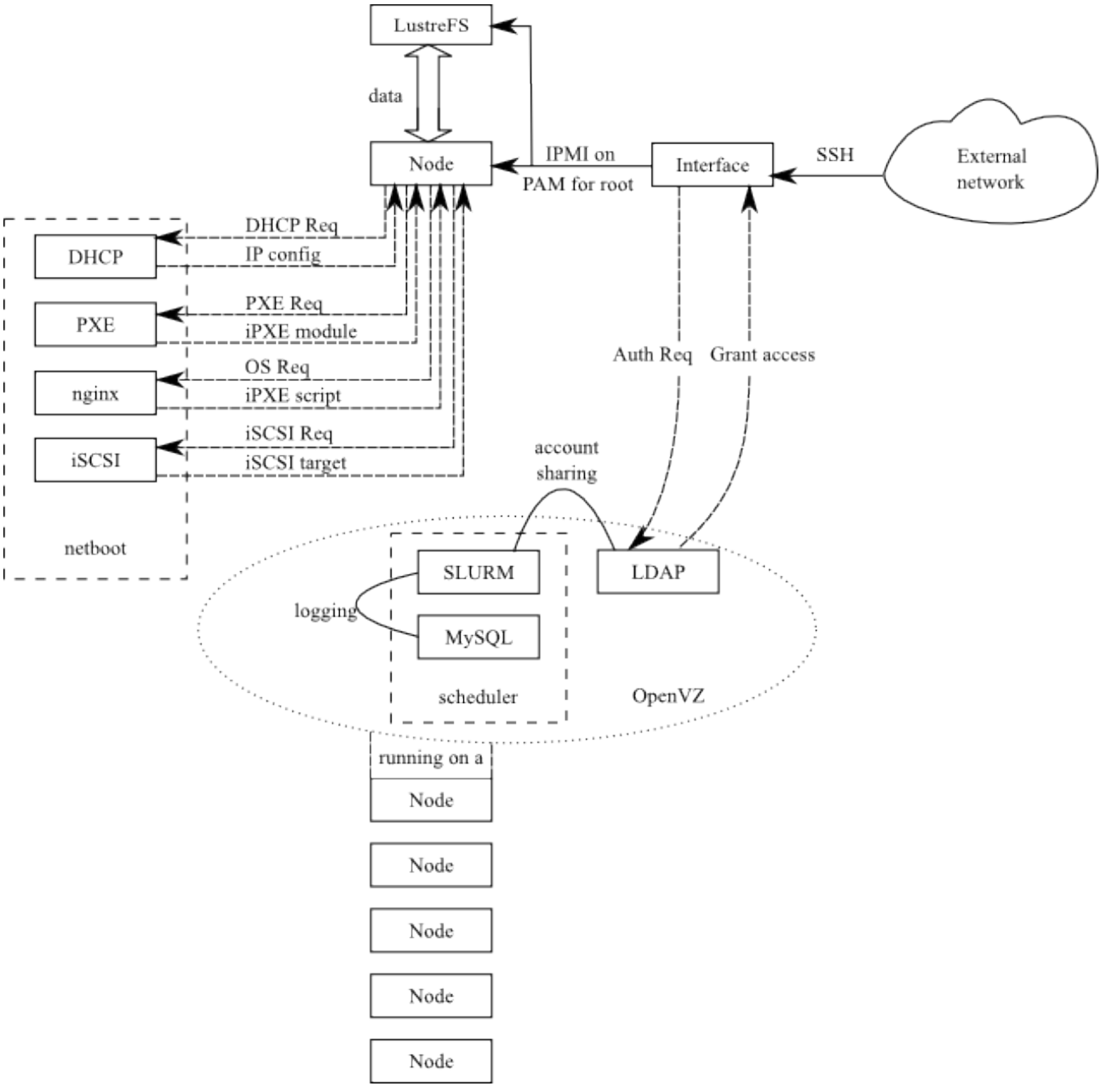

Figure 1. Cluster software architecture during the boot process

Furthermore, this dual-stage network boot approach has the following benefits. It allows to attach and detach computational nodes on demand, which is particularly useful to balance usage between available hardware or to do maintenance. In the latter case, the node to work with is put in the drain mode, which disallows scheduler to place any new tasks on it, and after than is shut down. It allows migrating services between nodes using

one of two available approaches. The first one consists in loading another node with from a services image and changing a DNS entry, which may result in an service interruption. The second one requires to virtualize services as proposed above, in which case the virtualization-enabled image is loaded on another node and then virtual machines, providing services, are migrated to this newly booted node with almost no delay in service.

Since most clusters use several network connections, typically at least one for control and another for data communication, the latter connection being faster, it might be used for the initial cluster booting: on this stage, no data required for user computations is transferred over this communication.

In order to speed up the boot sequence in large clusters, multiple network boot servers can be used, for example one server per server rack. In this case, operating system images should be synchronized between the network boot servers before the load. One can easily design a hierarchical sequence, in which the images are distributed through the hierarchy and only the last level in is used in a network boot procedure.

3 Dual-layer software

While the benefits of using read-only root disk are argued above, user home directories can remain read-only. To address this issue, we exploit the fact that the large storage is typically provided by a separate group of servers or specialized hardware and the nodes are diskless. User home directories are created on a network-attached storage device, using Lustre filesystem in our case. The attachment is performed to any computational node to allow users and their programs to transparently access the files from any place. This connection can also utilize specialized connection, e.g. Infiniband, to speed up access.

Another major drawback of read-only images boot over the network is that popular computational software tend to have large size, which might significantly increase the time required to transfer the image over the network while all the software is rarely used simultaneously on the same node. Also, several packages might require configuration tuning, which is impossible with read-only images. This issue can be addressed by moving most userspace software to the special partition of the network file system. Only the key system components, like drivers, network utilities and text editors, which are not susceptible to change often, can remain in the read-only image.

Such software layering also provides certain security guarantees: third-party software is installed separately and can't alter operating system installation or configuration. As an additional bonus, this facilitates the use of floating-licensed software provided license server runs as a separate service.

To ensure system security, which is quite important in multiple-user systems, software must be kept up to date. The dual-layer software approach we propose requires separate ways of managing software.

Core software, integrated in the bootable image, is updated from official distribution repositories following the normal procedure. To enable these updates, the special versioning script is used. By default, three last images are kept as is and all the others archived. The archives can be manually deleted. One can gain in used space by applying various incremental storage schemes. The images are enumerated as iSCSI targets when the network boot server starts. To do any changes to the bootable image, we use a special read-write target, which can be loaded on a single node. Once this image loaded, it can be changed arbitrarily, including software updates. During the shutdown process, the modified image will be stored back to the network boot server. Our script allows to automate this process by supporting the following operation modes: - force iSCSI target re-enumeration, based on naming schema to catch up with the latest image changes; - rollback modifications by making a carbon copy of the last read-only image to the read-write one; - deploy modifications by making read-only image with the last version number from the read-write image, and archive the oldest available one. After the modifications deployed, the boot scripts can be updated to use the latest version. Thus, on the next reboot any node will use the latest version.

Core software installed on a cluster tend to have stable rarely updated versions. On the contrary, application programs are subject to regular improvements. A part of users typically prefer to use the latest available software versions and even making changes to the application programs themselves, which eventually requires rebuild and reinstallation. Another part of users stays attached to particular software versions to avoid compatibility issues. These differences justify the use of different approaches to version management for core and application software. To keep application software up-to-date, we use EasyBuild software management system, which allows to build software from sources regularly and to maintain different versions simultaneously thanks to environment modules. EasyBuild also allows us to build new versions of compilers and core language libraries regularly to enable users generating faster codes. Since different compilers or even different versions of the same compiler can produce codes, performance of which can vary by an order of magnitude in accordance to our studies, the default compiler is removed from the environment to force user to make the informed choice of the toolchain

used to build his program. Taking various toolchains into account (a toolchain typically comprises a compiler and an MPI implementation), one should build every application program or library with every toolchain, which might require a significant amount of time. To deal with this problem, we propose to use separate, possibly virtual, build machines to from a so called build farm. In exclusive cases, for example when a toolchain gets a critical update requiring to rebuild all the software available, several nodes can be rebooted to a services image to run build machines and transformed back into the normal computational mode afterwards. Since the application software is susceptible to regular changes, it is stored on the network file system along with the user home directories. As an extra benefit, this allows users to modify configuration or source codes of the application software, provided corresponding permissions in the system.

Environment modules system itself prevents environment from being cluttered by different versions of the same software as well as results in smooth software interaction experience despite having multiple version combinations.

4 Discussion

The software architecture that we propose in this paper can be easily implemented using only freely available software, which is the case of our running example in NTUU "KPI" High-performance computing center.

Unlike cloud computing, the proposed approach doesn't virtualize computation nodes in order to get better performance while performing computations. Nevertheless, it provides means of centralized management of both hardware and software, features elasticity via horizontal on-demand system scaling and hierarchical scalability and provides certain security and reliability guarantees. Furthermore, this approach can benefit from services virtualization for the parts that are not performance-critical.

Unlike traditional cluster system management, the proposed approach features device independence and allows for smooth change of tasks performed by a particular hardware node depending on the network boot configuration, which is controlled in a centralized way.

The proposed architecture also allows to manage power consumption by means of controlling hardware that is powered on for different tasks and to consolidate virtualized services on a single node in case of low activity.

In our reasoning we assumed that a cluster system is mainly utilized for research activities, which isn't subject to accounting and financial regulations. Nevertheless, the proposed system allows to perform accounting tasks by using SLURM accounts.

Conclusion

Our approach allows to abstract away the hardware used in the cluster system and provides a consistent centralized management paradigm. It features multiple widely-required features like scalability, multitenancy and reliability while delivering the maximum performance rates for computations. The proposed dual-layer approach for software management allows to fulfill most user requirements and, in combination with the dual-layer start up comprising read-only images, provides a decent level of system security since critical parts of the system can't be modified.

The further work in this direction should consolidate into a single unified management interface the approaches described in this paper and their implementations as separate scripts and programs. Communication with the grid infrastructure can be also studied in more detail.

References

[1] J. J. Dongarra: Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment. *Technology and Science of Informatics*, 3(5): 317-321, 1984.

[2] E. Gabriel *et al.*: Open MPI: Goals, concept, and design of a next generation MPI implementation. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*: 97-104, 2004.

[3] H. Sutter: The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), 202-210, 2005.

[4] H. Sutter, J. Larus: Software and the concurrency revolution. *Queue*, 3(7), 54-62.

[5] A. Gordon *et al.*: ELI: bare-metal performance for I/O virtualization. *ACM SIGARCH Computer Architecture News*, 40(1), 411-422, 2011.

[6] H-C. Rohland, F. Kilian: Cluster architecture having a star topology with centralized services. U.S. Patent No. 8,190,780. 29 May 2012.

[7] A. McNab, A. Forti: Integrated cluster management at Manchester. *Journal of Physics: Conference Series* 396(4), 42039-42040, 2012.

[8] A. Kudryavtsev, V. Koshelev, A. Avetisyan: Modern hpc cluster virtualization using kvm and palacios. In *High Performance Computing (HiPC), 2012 19th IEEE International Conference on* 1-9, 2012.

[9] H. Sutter: Welcome to the Jungle. *Dr Dobb's Journal*, 42(8), 124-133, 2011.

[10] S. V. Davidson, R. J. Peterson: System and method for computer cluster virtualization using dynamic boot images and virtual disk. U.S. Patent No. 8,190,714. 29 May 2012.