LLSuperCloud: Sharing HPC Systems for Diverse Rapid Prototyping

Albert Reuther, Jeremy Kepner, William Arcand, David Bestor, Bill Bergeron, Chansup Byun, Matthew Hubbell, Peter Michaleas, Julie Mullen, Andrew Prout, Antonio Rosa
Computing and Analytics Group
MIT Lincoln Laboratory
Lexington, MA, USA
{reuther, kepner, warcand, david.bestor, bbergeron, cbyun, mhubbell, pmichaleas, jsm, aprout, antonio.rosa}@ll.mit.edu

***Abstract*— The supercomputing and enterprise computing arenas come from very different lineages. However, the advent of commodity computing servers has brought the two arenas closer than they have ever been. Within enterprise computing, commodity computing servers have resulted in the development of a wide range of new cloud capabilities: elastic computing, virtualization, and data hosting. Similarly, the supercomputing community has developed new capabilities in heterogeneous, massively parallel hardware and software. Merging the benefits of enterprise clouds and supercomputing has been a challenging goal. Significant effort has been expended in trying to deploy supercomputing capabilities on cloud computing systems. These efforts have resulted in unreliable, low-performance solutions, which requires enormous expertise to maintain.**

LLSuperCloud provides a novel solution to the problem of merging enterprise cloud and supercomputing technology. More specifically LLSuperCloud reverses the traditional paradigm of attempting to deploy supercomputing capabilities on a cloud and instead deploys cloud capabilities on a supercomputer. The result is a system that can handle heterogeneous, massively parallel workloads while also providing high performance elastic computing, virtualization, and databases. The benefits of LLSuperCloud are highlighted using a mixed workload of C MPI, parallel MATLAB, Java, databases, and virtualized web services.

Keywords – virtual machines, cloud computing, high performance computing, .

I. INTRODUCTION

A. Enterprise clouds

The underpinnings of current enterprise cloud computing and general shared computing resources have been developing and evolving for over four decades. The first virtual machines were developed to share expensive mainframe computer systems among many users by providing each user with a fully independent image of the operating system. On the research front, MIT, Bell Labs, and General Electric developed the Multics system, a hardware and operating system co-design that featured (among many other things) virtual memory for each user and isolated program execution space [1]. Commercially, the pioneer was IBM with the release of System 360/67, which presented each user with a full System 360 environment [2]. This line of capability development enjoyed over two decades of improvements, but shared computing resources fell out of favor as x86-based personal computers and servers became more widely accessible and affordable at work and at home.

The x86-based platforms were not initially designed for virtualization, which is essential for shared cloud enterprise computing capabilities. However, the beginning of a resurgence of shared enterprise server systems started in 1997 when VMware developed a technique based on binary code substitution (binary translation) that enabled the execution of privileged (OS) instructions from virtual machines on x86 systems. Further strides were made with the Xen project, which in 2003 used a jump table for choosing bare metal execution or virtual machine execution of privileged (OS) instructions. Such projects prompted Intel and AMD to add the VT-x and AMD-V virtualization extensions to the x86 and x86-64 instruction sets in 2006, which further pushed the performance and adoption of cloud computing environments for shared enterprise computing solutions. As these technology developments gained acceptance, companies including Amazon, Salesforce.com, Google, Rackspace, and many others saw opportunities to productize shared enterprise computing offerings.

Modern virtual machines enable dynamic allocation of hardware resource assignments for virtual machines, and they enable live migration of running virtual machines from one hardware server to another. However, such resource management capabilities are not frequently utilized by most users; mostly the capabilities are used for planned server maintenance by highly skilled server system administrators. Further, the algorithms for allocating server resources are fairly simple.

B. Cloud Computing for HPC

Meanwhile in the high performance computing (HPC) industry, simultaneously sharing system time among many users was intuitively assumed from early on. Like one-of-a-kind national-asset telescopes, large supercomputing systems were designed and built to be shared among many researchers, often across many scientific disciplines. Even today, supercomputers are seldomly allocated to any one user for any significant stretch of time; the systems are almost always executing jobs for multiple users simultaneously. However, just

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

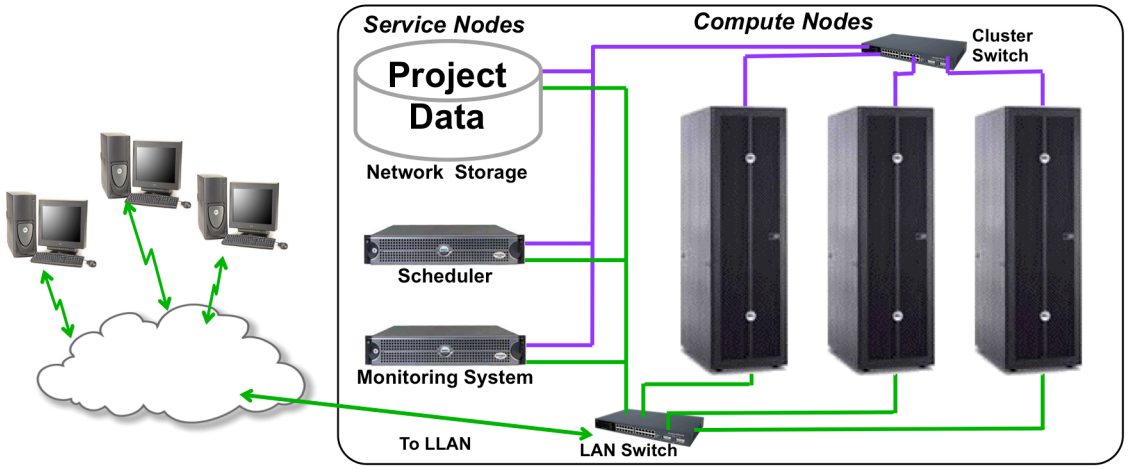

Figure 1: The LLGrid System Architecture

as with shared enterprise computing systems, the ubiquitous adoption of the x86-based computing platform made HPC solutions more widely accessible and affordable to more researchers and businesses. It should be noted that there are some subtle differences in the feature sets of shared enterprise x86-based servers and HPC servers, particularly in their network fabric and floating point computational capability.

A number of companies and projects have demonstrated shared enterprise cloud and HPC capabilities on the same solution platform. Most notable of these is Amazon Web Services, which offers HPC nodes and GPUs [3]. However, these solutions all require that all computations must be conducted within virtual machines, regardless of whether they can execute natively on the underlying hardware and software stack. This means that all cloud enterprise and HPC applications must incur the performance penalties, particularly in disk, network, and other I/O transactions [4].

C. *LLSuperCloud - shared HPC cloud capabilities*

Over the past ten years, the LLGrid [5] at MIT Lincoln Laboratory has evolved from a four-node prototype cluster running single-user MatlabMPI jobs to a constellation of systems serving well over 300 users each year. LLGrid was developed to enable the rapid prototyping computational needs of MIT Lincoln Laboratory, providing interactive, on-demand parallel and distributed simulation, data processing, and algorithm exploration and development capabilities across a wide range of DoD mission areas including ballistic missile defense, radar digital signal processing algorithm development, aircraft collision avoidance algorithm verification, communication channel reliability modeling, and satellite propagation simulations. With this goal in mind, the LLGrid team continues to explore novel ways to accommodate various high performance computing requirements and needs on shared HPC hardware systems.

The LLGrid system, depicted in Figure 1, is comprised of two primary types of nodes: compute nodes and service nodes. Among the service nodes, there is further distinction. The login nodes are compute nodes on which the scheduler is forbidden to launch jobs. The file system server nodes export the central file system to all of the other nodes in the cluster as well as user laptops and desktops. The management node is used to deliver the appropriate software stack to all of the nodes in the cluster based on each node's role in the cluster. Finally, the scheduler matches job launch requests (including resource requirements for each job) with available compute node resources. The scheduler also reports on the status and resource usage of jobs, and keeps track of the system status of all of the compute nodes in the cluster. Currently, LLGrid uses Open Grid Scheduler [6], but LLGrid has also run on the Condor [7], OpenPBS/TORQUE/PBS Pro [8,9], and LSF [10] schedulers.

It is the centrality of the scheduler that enables many different types of jobs to execute simultaneously on LLGrid including interactive, on-demand parallel MATLAB jobs; parallel C/MPI, C++/MPI, Java, and Fortran jobs; MapReduce jobs; dynamically-allocated databases; and virtual machines. Regardless of the job type, they are all processes that are brokered and launched on compute nodes using the scheduler's remote shell. After all, a scheduler is just a sophisticated cluster resource manager, which determines what resources should be used to execute one or more remoted shell-based executable. Once resource assignment has been established, the scheduler launches the jobs using its remote shell infrastructure. In the next several sections, we will discuss how each of these different types of capabilities are enabled.

II. pMATLAB AND gridMATLAB JOBS

The establishing goal of the LLGrid project was to develop a on-demand, interactive grid computing capability for MATLAB – the dominant programming language for implementing numerical computations, widely used for algorithm development, simulation, data reduction, testing, and system evaluation – as its initial target application. Choosing MATLAB was simple back then: Lincoln had over one thousand MATLAB users; nearly two hundred users run very long jobs that could benefit from parallel processing. (Today those numbers are higher.) The LLGrid project had developed three technologies that allow these users to run parallel MATLAB jobs transparently on the LLGrid systems:

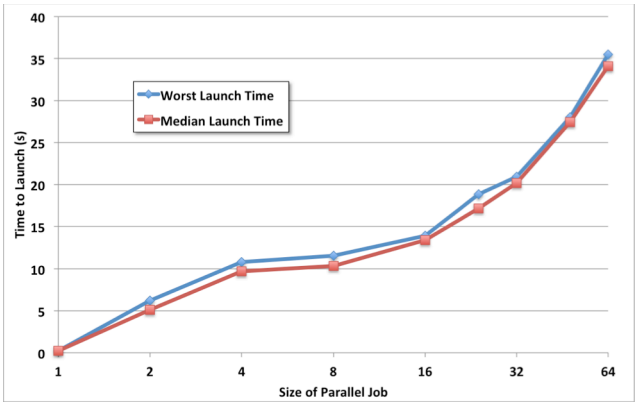

Figure 2: Launch time latency for pMatlab job launches on LLGrid.

- *MatlabMPI* for point-to-point messaging [11];
- *pMatlab* for parallel global array semantic (PGAS) programming [12,13]; and
- *gridMatlab* for integrating user's computers into the LLgrid and automatically allocating grid computing resources [14].

These three toolboxes are combined to create a unique on-demand, interactive grid computing experience, whereby running a parallel MATLAB job on LLGrid is identical to running MATLAB on the desktop. Users can use LLgrid from Windows, Linux, and Mac OS X computers with their desktop computer becoming a personal node in the LLGrid thereby establishing a transparent interface between the user's computer and the grid resources. LLGrid is enabling faster algorithm development, prototyping, and validation cycles for Lincoln staff.

Further description of the gridMatlab toolbox is merited with regard to building a multi-purpose shared cloud computing capability. The gridMatlab toolbox transparently integrates the MATLAB on each user's desktop with shared grid clusters through the cluster scheduler by launching a job array of parallel MATLAB processes whenever a MatlabMPI or pMatlab job is run by the user in his or her MATLAB session. These parallel MATLAB processes join the user's MATLAB session. Similarly, job status checks, LLGrid system status checks, and job aborts are also issued with MATLAB commands that call the scheduler to perform the requested action. Essential to enabling interactive, on-demand parallel activities is low latency launch times. Figure 2 shows the launch times for pMatlab jobs from one to 64 processes per job. As one might expect, users' expectations for job launches depends on how many parallel processes they are launching, and we have found that users are consistently content with the launch times that the system delivers. However, users are quick to request help when job launches slow down. The LLGrid team has found this advantageous because slow launch times are often a symptom of some underlying system issue.

III. Batch Computing Jobs and LLMapReduce Jobs

For many applications and simulations at Lincoln Laboratory, interactive, on-demand launching (particularly MATLAB) is the ideal environment. However, there are other users who have many single process jobs, many independent single process jobs (job arrays), or parallel (one job with many communicating simultaneous processes) jobs. These jobs are in any of a number of programming languages including C/MPI, C++/MPI, Fortran, Java, Python, MATLAB, etc. For regular batch jobs, whether single processes, job arrays, or parallel jobs, the user interface is the command line on login nodes.

In recent years with the gaining acceptance of the Hadoop [15] framework, the map/reduce programming model has gained familiarity and popularity. The map/reduce parallel programming model is the simplest of all parallel programming models, and it is arguably much easier to learn than message passing or distributed arrays. The map/reduce parallel programming model consists of two user-written programs: the Mapper and the Reducer, each of which have input and output files. The output of the Mappers is the input of the Reducer; there are one or more Mappers, but only one Reducer. Launching consists of starting many Mapper programs each with a different file, and when the Mapper programs have completed the Reduce program is run on the Mapper outputs to aggregate or consolidate the results.

LLGrid MapReduce enables map/reduce for any programming language using a simple one-line command. Upon launching such a job, LLGrid MapReduce identifies the input files to be processed by scanning a given input directory or reading a list from an input file. It then submits a job array with one process for each file, and the user can choose how many of these job array processes are executed concurrently. Once all of the input files are processed by the Mappers, there is an option to collect the results using the Reducer. The reduce task will wait until all the mapper tasks are completed by setting a job dependency between the mapper tasks and the reducer task. LLGrid MapReduce is covered in more detail in [16].

IV. D4M and Accumulo - Dynamic Databases for HPC

Scientific computing jobs and virtual machines can be I/O intensive, but many database instances typically execute in a very I/O intensive manner. This is because the contents of database tables often cannot be contained in their entirety in computers main memory. Hence, database administrators (DBAs) implement database instances using a local hard drive or locally attached disk arrays; most DBAs would not think of hosting databases on shared network storage. Also, most databases are considered network data services shared by many users or used by applications that are used by many users, so they are usually turned on and not brought down or moved except for occasional maintenance.

Databases used for scientific computing and data research (Big Data) are often different. The requirements for their use still require intense I/O capability, but they are often used by only a single user (or a handful of users), and they usually used in spurts; i.e., they are used intensely to do some data ingests, searches, and analyses but then are not used for some stretch of days or weeks until the next intense session. Such a usage pattern lends itself well to using shared HPC servers and a scheduler to manage on which server the database instances run. But there are a few further challenges: using a consistent

URL for the database service and guaranteeing acceptable I/O performance for the database instance. The first challenge has been overcome by deploying a custom database-backed dynamic DNS (domain name service) capability; this LLGrid-hosted service has a RESTful web service interface and is available throughout the Lincoln local area network (LLAN).

But providing acceptable I/O performance for database instances is more challenging. Running the database from a data store on the central file system delivers subpar performance due to the network latency from the compute nodes to the central storage – databases are just too closely knit to the underlying disk storage. So for the LLGrid dynamic databases solution, another option was developed. The path location of the database datastore in the file system has been standardized, as is the path location of where the database datastore is archived on central storage. LLGrid database instances can be managed from the login node command line or through a web browser interface. When a new database is instantiated, the scheduler dispatches the database launching script to one of the compute nodes. Once launched, the database creates its data files in the temporary storage area of the local disks of the compute node. When the database instance is suspended, a suspension script shuts down the database instance on the compute node, zips up the database data files into an archive file and copies the database data archive onto the central storage. When the database instance is re-instantiated, the scheduler finds a new compute node on which to launch the database instance, the database archive file is copied to that compute node and unzipped, and the database process is launched again. The current LLGrid dynamic database capability supports Apache Accumulo [17] NoSQL database. Table 1 shows the typical launch times for several scenarios. All of these timings were taken with Accumulo databases.

Table 1: Execution times for dynamic Accumulo database startups and stops.

Scenario	Execution Time
Empty database startup	~90 sec
Empty database stop	~90 sec
13.6 GB database startup	~240 sec
13.6 GB database stop	~90 sec
200 GB database startup	<10 min

Currently, LLGrid dynamic databases can only be single process / single node instances of a given database, but if the demand exists, clustered databases could also be implemented with this dynamic capability.

V. VIRTUAL MACHINES FOR HPC

For LLGrid users, the two most appealing advantages of using virtual machines are rapid distributed computing system prototyping and support for legacy operating systems. Often a part of starting a new research project involves purchasing one or more servers before really understanding the computational requirements of the application codes. Users can configure and execute VM instances on LLGrid so that their research projects can explore the computational requirements of their application codes without having to wait for a server order to arrive. With regard to supporting legacy operating systems, we routinely work with research teams that have a scientific code suite that has been validated and verified with a certain OS type, version, etc. that is different than the one installed on LLGrid. Usually this means that the research team must revalidate and re-verify their scientific code suite on the LLGrid software stack. Another scenario is that the scientific code suite is no longer supported on the hardware and OS that comprises LLGrid. If they could execute their code suite within virtual machines with the identical environment in which they were verified, the research teams would sacrifice some performance, while gaining productivity by not having to revalidate and reverify.

The abstraction layer on which virtual machines are executed is called the hypervisor, and there are two types of hypervisors. Type 1 hypervisors are installed directly on the hardware; in this deployment, the hypervisor is essentially a minimal OS for executing virtual machines. Type 2 hypervisors are installed within a host operating system, and they are executed as a process in the host OS. Each guest virtual machine is then a child process to the hypervisor process, and processes within each virtual machine are co-managed by each virtual machine OS and the hypervisor. Because the hypervisor and virtual machine processes are just common processes, they can be launched through the cluster scheduler, which is how we handle virtual machine jobs on LLGrid [18]. We have demonstrated this capability with VirtualBox [19] and VMWare [20] hypervisors and virtual machines, and we intend to demonstrate it with some other VM types.

To enable fast launch times, we stripped down the VM operating system image to have no services and only the libraries that were required for scientific computing. If other libraries are needed they will be symlinked into the users virtual machine environment from a centrally shared library repository. The job submitted to the scheduler clones a VM image, registers the image to the compute node that the scheduler has chosen, and begins the boot sequence of the VM. The job execution in the VM is written into the initialization scripts of the VM so that when the VM has fully booted, it immediately begins execution of the job. After execution of the job, the VM enters the shutdown script, and upon shutdown the VM image is discarded. If the VM job is aborted before execution is completed, the VM catches the kill signal sent by the scheduler, and the VM is cleanly powered down and discarded. While the default is to launch one VM job per job slot, we have added the ability to overload each scheduler jobslot with multiple VMs. The resources that are available on a compute node limit the number of VMs that can be launched per jobslot.

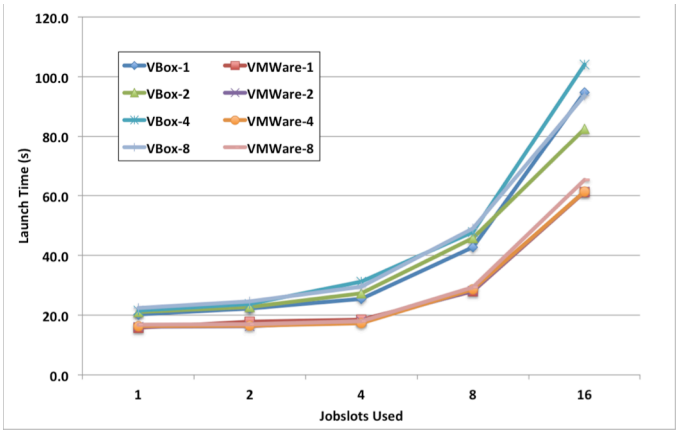

Figure 3: Virtual machine launch times on LLGrid.

Figure 3 shows the launch times of optimized VM instances for both VirtualBox and VMWare hypervisors and VMs. VMWare is faster both at the low end (one VM on one node) and the high end (16 VMs on each of 8 nodes, 128 total) – between 25% and 35% faster across number of jobslots and VMs per jobslot. Further, VMWare did not have the reliability problems that VirtualBox did. The VMWare launch scripts ran straight through with no failures – with VirtualBox there were occasional deadlocks that required some manual cleanup while running the benchmarks.

VI. Scheduling Jobs

Resource management and scheduling on the LLGrid systems have evolved over the past few years. Early in the project we ran LLGrid exclusively as an interactive, on-demand computing resource. This worked well for the most part, until we opened up access for accommodating batch-style jobs. As expected, our system was swamped by several large batch-style jobs, and no interactive jobs could be run for hours.

Currently LLGrid utilizes a hybrid scheduling policy [21]. This policy enables two queues, one for interactive, on-demand jobs and a second for batch jobs. The interactive, on-demand parallel MATLAB jobs, dynamic database, and virtual machine jobs are launched through the interactive, on-demand queue, while the batch and LLGrid MapReduce jobs are launched through the batch queue. The interactive, on-demand queue has a higher priority on jobs slots (CPUs) than the batch queue. However, the interactive, on-demand queue has a lower per-user CPU limit of only 64 CPUs per job (more upon request) versus the 48 or 96 (day/night) CPU limit on the batch queue. On future LLGrid systems that have more CPU cores/jobslots, these limits will rise.

VII. Related Work

In the ten years that LLGrid has been evolving many other organizations have adopted interactive HPC solutions. Some have implemented this using the slot reservation systems available with many modern HPC schedulers, while others have provisioned extra nodes and job scheduling policies like LLGrid. Batch computing and MapReduce jobs are common on almost all HPC systems. Hadoop [15] has a capability for dynamically launching Hadoop environments, but this generally refers to HDFS and MapReduce capabilities; HBase is always statically deployed. Also, for many years, the IBM Platform Symphony product has been utilized for launching and managing web services that front databases in an instantaneous on-demand fashion [22].

However, scheduling databases and virtual machines as HPC jobs is a rather unique endeavor. We are not aware of any other organization that is dynamically launching and managing shared database instances. However, there are a few instances of research with launching virtual machines onto HPC systems.

A team at Taiwan's National Center for High-Performance Computing, Tainan, developed a hybrid bare-metal/virtualized computing cluster called Formosa3 [23]. They modified the Torque scheduler to interface and manage OpenNebula, which subsequently managed the VMs on the compute nodes (an approach 1 and 2 hybrid with type-2 hypervisor). OpenNebula was responsible for provisioning and deprovisioning VMs on the compute nodes; once a compute node was provisioned, it signaled Torque that the VM was ready to accept a VM job. In their cluster, compute nodes could either run VM jobs or HPC jobs, but they did not allow the two types of jobs to comingle on the same compute node. They included benchmarking of VM job launches and found similar results to our unoptimized results.

The Clemson University School of Computing team used the KVM hypervisor on a cluster for grid computing research [24]. They statically allocated VMs onto compute nodes with the hypervisor running on the compute node operating system (with Type 2 hypervisor), and they used the Condor scheduler to launch jobs onto the static VMs. They demonstrated how a virtualized cluster could be a part of a multi-organization grid computing infrastructure and ran a variety of benchmarks. Just like the Formosa3 team, they used stock OS images.

VIII. Summary and Future Work

This article has explained how a variety of different prototyping job types are being accommodated on the LLGrid shared HPC cluster computing system. Particularly, we have explored how interactive, on-demand jobs, batch jobs, map/reduce jobs, dynamic database jobs, and virtual machine jobs are launched through a common cluster resource manager/scheduler. By enabling such varied application technologies, the LLGrid system is the most versatile computational platform for research computing.

In the future, we plan to extend the dynamic database capability to add dynamic clustered/distributed database support. Also we will continue to add capabilities to the virtual machine offerings by including virtual network support and parallel virtual machine job launches for legacy MPI applications.

References

[1] F.J. Corbató and V.A. Vyssotsky, "Introduction and overview of the Multics system," AFIPS, 1965.

[2] R. Dittner and D. Rule Jr., *Best Damn Server Virtualization Book Period*, Syngress, 2007.

[3] High Performance Computing – Amazon Web Services, http://aws.amazon.com/hpc-applications/.

[4] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V. Weaver, J. Dongarra, "Evaluation of the HPC Challenge benchmarks in virtualized environments," *6th Workshop on Virtualization in High-Performance Cloud Computing*, Bordeaux, France, August 30, 2011.

[5] N. Bliss, R. Bond, H. Kim, A. Reuther, and J. Kepner, "Interactive grid computing at Lincoln Laboratory," Lincoln Laboratory Journal, vol. 16, no. 1, 2006.

[6] OpenGridScheduler, http://gridscheduler.sourceforge.net.

[7] High Throughput Computing Condor, http://research.cs.wisc.edu /htcondor/.

[8] TORQUE Resource Manager, http://www.adaptivecomputing.com /products/open-source/torque/.

[9] PBS Professional, http://www.pbsworks.com/.

[10] IBM Platform LSF, http://www-03.ibm.com/systems /technicalcomputing /platformcomputing/products/lsf/.

[11] J. Kepner and S. Ahalt, "MatlabMPI," Journal of Parallel and Distributed Computing, vol. 64, issue 8, August, 2004.

[12] N. Bliss and J. Kepner, "pMatlab parallel Matlab library," International Journal of High Performance Computing Applications: Special Issue on High Level Programming Languages and Models, J. Kepner and H. Zima (editors), Winter 2006 (November).

[13] J. Kepner, Parallel Matlab for Multicore and Multinode Computers, SIAM Press, Philadelphia, 2009.

[14] A.I. Reuther, T. Currie, J. Kepner, H.G. Kim, A. McCabe, M.P. Moore, N. Travinin, "On-Demand Grid Computing Using gridMatlab and pMatlab," In *Proceedings of the High Performance Computing Modernization Office Users Group Conference 2004*, Williamsburg, VA, 8 June 2004.

[15] Apache Hadoop, http://hadoop.apache.org/.

[16] C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, J. Kepner, A. McCabe, P. Michaleas, J. Mullen, D. O'Gwynn, A. Prout, A. Reuther, A. Rosa, and C. Yee, "Driving Big Data With Big Compute," IEEE High Performance Extreme Computing (HPEC) conference, Waltham, MA, September 10-12, 2012.

[17] Apache Accumulo, http://accumulo.apache.org/.

[18] A. Reuther, P. Michaleas, A. Prout, and J. Kepner, "HPC-VMs: Virtual Machines in High Performance Computing Systems," IEEE High Performance Extreme Computing (HPEC) conference, Waltham, MA, September 10-12, 2012.

[19] Oracle VM Virtual Box, https://www.virtualbox.org/.

[20] VMWare Virtualization, http://www.vmware.com/.

[21] A. Reuther, J. Kepner, A. McCabe, J. Mullen, N.T. Bliss, and H. Kim, "Technical Challenges of Supporting Interactive HPC," In *Proceedings of the High Performance Computing Modernization Office (HPCMO) Users Group Conference (UGC) 2007*, Pittsburgh, PA, 18-22 June 2007.

[22] IBM Platform Symphony Software, http://www-03.ibm.com /systems/technicalcomputing/platformcomputing/products/symphony/.

[23] C.H. Li, T.M. Chen, Y.C. Chen, and S.T. Wang, "Formosa3: A cloud-enabled HPC cluster in NCHC," World Academy of Science, Engineering, and Technology Journal, Vol. 73, No. 38, 2011.

[24] M. Fenn, M.A. Murphy, S. Goasguen, "A study of a KVM-based Cluster for Grid Computing," In *Proceedings of the 47th Annual Southeast Regional Conference* (ACM-SE 47). ACM, New York, NY, USA, Article 34.