ABSTRACT

TALPALLIKAR, NIKHIL VIVEK. High-Performance Cloud Computing: VCL Case Study. (Under the direction of Dr. Mladen Vouk.)

High performance computing used to be domain of specialties and of the relatively few. Today, as internet has become pervasive and large amounts of data are pouring in, extensive analyses of the data and simulations are gaining more importance in decision making. This fuels use of high-performance computing (HPC), more recently high-performance data (HPD) analytics in everyday workflows. For example, advanced internet searches invariably involve loosely coupled high-performance computing, large scale business analytics also requires high-performance facilities. Security analytics and large-scale scientific analytics are hard to imagine anymore without involving an HPC/HPD platform. Properly constructed clouds should be able to offer the needed services on demand.

This project examines the feasibility and practice of offering a full range of HPC/HPD services (including possibly supercomputing services) in a cloud environment. The specific environment in which this was investigated is called VCL (http://vcl.ncsu.edu). It is a production-level cloud computing solution serving about 40,000+ users at NC State University. VCL, in addition to general purpose services such as desktops and servers, offers HPC services. In this work, of special interest is provisioning of HPC in the cloud and the performance such resources offer. For example, how easy it is to integrate general purpose graphical processing units (GPGPU) into Linux clusters and then offer them as a workflow option to cloud users? As part of this work we developed a test-bed based on VCL to study such questions. This test-bed includes general VCL services such as desktops and gateways, HPC production level facilities, an experimental GPU cluster called ARC, and VCL HPC queues that offer pre-configured HPC cluster services. Control of cluster interconnect topologies and speeds is of special interest since this can dictate performance. Architecture of different approaches and solutions is described and benchmarked to illustrate the skill and knowledge-level required for set-up and use of different solutions. This includes benchmarks

such as, High Performance LINPACK, Ping-Pong test which is a part of the HPC Challenge (HPCC) suite of benchmarks and a real-world high-performance scientific application.

Results indicate that a properly pre-configured and tuned cloud-based HPC cluster can achieve very good performance. On the other hand, on the fly constructed clusters using cloud resources may operate well for loosely coupled applications, but may offer poor performance in the case of tightly coupled applications (HPC). This is primarily due to the latency on message passing interconnects (such as those carrying MPI traffic) unless it is explicitly controlled by the end-user. In a cloud environment, it may be difficult to guarantee that similar machines are provisioned each time new resources are requested. Therefore the application performance may vary every time an HPC/HPD service is requested. We illustrate this in the context of VCL and discuss solution options available to cloud HPC users.

High-Performance Cloud Computing: VCL Case Study

by
Nikhil Vivek Talpallikar

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Science

Raleigh, North Carolina

2012

APPROVED BY:

_____	_____
Dr. Vincent Freeh	Dr. Rudra Dutta

_____

Dr. Mladen Vouk

Committee Chair

DEDICATION

To my parents, my elder sister Shruti and to all my friends

BIOGRAPHY

Nikhil Talpallikar was born in Hyderabad, India. He did his primary and secondary schooling from St. Joseph High School, Solapur, India. He received his Bachelors Degree from University of Pune in 2008. He worked at IBM Systems and Technology Group for two years in the area of SAN Virtualization. He joined the Masters Program in Computer Science at North Carolina State University in Fall 2010. He plays for the NC State Cricket Club as an opening batsman and wicket-keeper. His hobbies are reading books, playing soccer, cycling and trekking.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude towards Dr. Mladen Vouk for providing me the help and guidance not just through the course of my thesis but throughout the course of my graduate program at North Carolina State University. I would like to thank Dr. Rudra Dutta and Dr. Vincent Freeh for serving on my thesis committee.

I am highly obliged to Dr. Gary Howell for clearing my doubts regarding the intricacies of Henry2 cluster. I would like to take this opportunity to thank David Fiala and Yongpeng Zhang for helping me while working on the ARC cluster.

This work was supported in part by the DOE SciDAC grant DE-FC02-07-ER25484, by the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative, NSF grant #0910767, and by the IBM Shared University Research Program

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Thirst for processing power is growing. It is driven by decision making needs, large amounts of data, analytics, simulations, and entertainment – in the research domain some examples are bio-informatics, life sciences, electronics (magnetic and electric field simulations), finance (computations for market simulation), fluid dynamics, data mining, analytics, databases, computer vision and graphics, medical imaging, quantum physics/dynamics, weather forecast simulations, geographical modeling of land and oceans, security, and similar. High-performance computing (HPC) is going mainstream – even though many do not realize that. After all, most of the high-power searches are running on platforms and on the numbers of processors that few years ago would have been considered supercomputing facilities [44]. In the last few years, one of the innovations in that domain has been the move of general purpose graphical processing units (GPGPU) into more general computing space and use of their high compute power capability to solve computational problems that exhibit high data parallelism [74,75]. GPUs come in many forms and shapes, but in most cases as add-on PCI (Peripheral Component Interconnect) cards. Traditional HPC, range from Linux clusters interconnected by high performance communication channels, to genuine, specially built supercomputers such as Cray machines [95], IBM BlueGene [93] and similar. Today, commodity hardware using just CPUs features prominently on the Top 500 list [44]. In fact, technology advances in microprocessors (e.g., multicore chips, GPU cards), network interconnects and memory technologies have brought the cost of building a high performance computer down to acceptable levels. Still, the cost of HPC does not really allow most individual users, or even companies, to have that option on their desktops or in the organization, unless those resources are shared with others.

One approach of managing that barrier has started emerging over the last 5 years – cloud computing. Cloud computing allows elastic on-demand sharing of resources. In a properly designed environment a cloud subsystem can provide access to HPC facilities. Cloud

environments are flexible in terms of new resource additions and on demand utilization. Cloud computing provides for an abstraction which is more flexible option than the traditional in-house HPC clusters. Cloud computing seems almost ideal for provisioning services that range from desktops all the way to sub-clouds and full scale HPC facilities. On the other hand there are questions related to the ability of clouds to provide adequate HPC facilities [27,30,31,32,36,37,35,43,45,47]. Most of these arguments are based on the Amazon HPC services [42]. A few arguments equate virtualization with cloud computing [34] which, of course is not true, and as a result may have an incomplete view of what a properly constructed cloud can actually do when the model includes so called hybrid computing option[42,76, 79, 50].

What is out there?

HPC in cloud is a long and interesting debate [77, 78, 79] and research area. The Amazon EC2 [1] has extended its IaaS (Infrastructure as a Service) to provision HPC and GPGPU clusters since July 2010 [32, 42], and NC State's VCL cloud has been delivering HPC as a service since 2004 [82]. The Nimbus Project at University of Chicago and Indiana University is a similar service [83], and Beowulf clusters [84] running flavors of unix (Linux) also can provide cloud-like infrastructure for parallel computations.

As already mentioned, a lot of research in the area of HPC in the cloud has focused on the Amazon EC2 High Performance Clusters [27,30,31,32,36,37,35,43,45,47]. However, one has to remember that solutions such as Eucalyptus [80] and OpenStack [81], and in general many Hadoop platforms [71], also offer HPC in some form. HPC in the cloud can be focused on tightly coupled applications [27,30,31,36,41,35,47,48,49], and on loosely coupled environments (e.g.,Hadoop [71], distributed web-services, scientific workflow applications [37]). Different benchmarks have been used to analyze the performance of HPC in cloud. For example, J. Napper et al. [41] used the LINPACK benchmark to evaluate whether a cloud (Amazon EC2 [1]) based HPC solution can match the ranks of the top 500 supercomputing infrastructures.

Yan Zhai et al. in [27] evaluated the Amazon EC2 HPC resources in terms of overall performance, scalability, cost-effectiveness, and stability. They show that HPC in a public cloud, such as EC2, may not perform better than a dedicated in-house cluster. The main reason tends to be the communication channels (or node interconnects). EC2 uses 10Gbps Ethernet for that, while the in-house cluster may use a 40 Gbps QDR InfiniBand network. Of course, in house clusters with less communication bandwidth may not do so well. Research [27] shows that as the number of processes in a job increases the performance of EC2 cluster degrades. This is because the communication overhead increases as the number of processes increases. Also, the specialized QDR IB in-house network has less latency than the 10Gbps Ethernet on Amazon EC2 cluster [27]. There were also some performance stability issues with Amazon EC2 HPC clusters. Where Amazon EC2 appears to win, is in cost effectiveness. The average utilization of the cloud resources is higher as compared to the in-house cluster.

Another interesting study is that reported in [50]. The Nimbus cloud at the University of Chicago and Indiana University [83, 50] explores the option of extending the capacity of local in-house compute clusters by requesting on demand resources from a public cloud. They use Torque as the primary resource manager. As demand for compute power increases, new resources (virtual machines with required libraries and a Torque client or pbs_mom daemon preinstalled and preconfigured) are requested from the cloud and are added as additional compute nodes to the Torque database. A new job can now spawn threads to this new compute node, thus achieving flexibility. The performance characteristics of these hybrid clusters are not discussed in the paper with respect to compute intensive jobs. Instead, the focus is on the efficiency with which the requested extra resources join the original cluster. Cost-effectiveness is achieved since the job submission queue and requested wall time of the jobs is monitored. Since the work focuses on talking to the resource managers, virtual machines based on any hypervisor should work, as long as these compute resources are managed by a resource manager like Torque, Nimbus or EC2. New resources are requested by calling the API functions exposed by Torque, Nimbus or EC2. Likewise, VCL API can be called to integrate VCL with this elastic HPC in cloud solution.

The following provides an overview of a non-comprehensive but reasonably representative sample of commercial on-demand HPC offerings in the cloud.

Amazon Web Services (Elastic Cloud 2)

"Amazon Web Services (AWS) [1, 42] provides affordable, flexible and on-demand compute nodes for compute intensive applications. There is flexibility to add new nodes as well as remove some nodes. HPC nodes are grouped as special high end machines which are provisioned on demand from the HPC Placement group. The nodes use 10Gbps Ethernet interconnect for MPI traffic. The AWS HPC site [1] states that "By leveraging the advanced networking and high computational capabilities of Amazon EC2 Cluster instances, customers can provision clusters that can give them supercomputing class performance without the need to build and operate their own HPC facility. For example, a 1064 instance (17024 cores) cluster of *cc2.8xlarge* instances was able to achieve 240.09 TeraFLOPS for the High Performance Linpack benchmark, placing the cluster at #42 in the November 2011 Top500 list"[1].

"Amazon EC2 also has a system built for provisioning GPGPU clusters. EC2 provides high bandwidth, low latency networking, and very high compute capabilities via their Cluster Compute or Cluster GPGPU instances. EC2 provides for flexible Placement Groups where each Placement Group is a tightly coupled resource cluster and offers 10Gbps full bisection network bandwidth [1]. Each GPGPU cluster can be a part of one Placement Group or multiple placement groups depending on how massively parallel the jobs are. EC2 GPGPU provisioning provides for max 2 GPU units per node. The configuration is as follows : "Cluster GPU Quadruple Extra Large 22 GB memory, 33.5 EC2 Compute Units, 2 x NVIDIA Tesla "Fermi" M2050 GPUs, 1690 GB of local instance storage, 64-bit platform, 10 Gigabit Ethernet"[1].

"EC2 Compute Unit (ECU) – One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Although there is flexibility in cluster resources, there is no flexibility in terms of GPU instances. With regards

to performance an 880 instance (7040 cores) cluster of *cc1.4xlarge* instances was able to achieve 41.82 TeraFLOPS"[1]. More information can be found at [1, 42].

Nimbix

"Nimbix [7] delivers a scalable environment that extends the processing capabilities of existing high performance computing infrastructures using FPGA's and GPU's. Their computing infrastructure delivers accelerated hardware and applications that process data faster and more efficiently for industry specific tasks. Across multiple application domains, scientists and researchers are challenged by data processing requirements. Nimbix enables an easy to use service to accelerate high performance computing applications." [7] The Nimbix site [7] quotes "NACC offers on-demand processing for HPC applications using state of the art computing infrastructure. After setting up a Nimbix account here on the portal, you may launch secure high performance computing jobs using any of a number of common HPC applications. No need to provision cloud instances or manage systems. Simply upload data, select application and submit your job. You pay only for actual processing time." [7] This indicates that Nimbix offers pre-built common applications that run in HPC environments. The user has no control over the algorithm or code that the application is running. Nimbix offers to schedule common HPC jobs but on user provided data. This service is not a flexible service. The jobs might be running on in-house HPC clusters and there is no concept of an on demand service in this case.

PEER 1 Hosting

"PEER 1 hosting [8] use the NVIDIA's Tesla™ GPUs [14] along with the existing clusters and combine the advantages of private cloud computing to deliver affordable High Performance Cloud Computing (HPCC). Each GPU delivers a peak performance between 2 to 4 TeraFLOPS [8]. GPU servers are dedicated exclusively to users, there is no sharing of resources. Each server delivers up to four teraflops of single-precision peak performance and 515 gigaflops of double-precision performance [8]. Peer1 use NVIDIA Tesla M2050 GPU cards which have a peak performance of 515 GFLOPS with double precision floating point computations and 1 TFLOPS with single precision floats" [8].

Hoopoe

Hoopoe [9] web services provide for GPU computer power. “A front end web service interface is used to communicate with the world and perform the various operations available by Hoopoe. Applications submit tasks, monitor activities, gather statistics, etc via the web interface. Users are not granted direct access to cluster machines, but the web service front end provides all necessary tools and features to communicate with it, perform computations, read results etc. Hoopoe's GPU cluster is composed of 10's of GPU devices, delivering more than 50 TFLOPS of peak performance. As far as internal communications is concerned, the cluster is capable of delivering over 40 Gb/s using fast InfiniBand interconnect between the nodes” [9]. Again, this model is less flexible since the user has limited access to the actual cluster.

There are various other small start-ups emerging in this domain [52, 53]. Most of the new comers are trying to copy the Amazon EC2 architecture with some smaller differences.
Following are some of the open source cloud computing environments

Eucalyptus

“Eucalyptus [80] is a widely used software platform which provides for private cloud resources using the existing infrastructure. It is widely accepted to serve the use case of Infrastructure as a service (Iaas) for provisioning private and hybrid clouds. Eucalyptus provides for an abstraction over the existing machines in the lab. This abstraction enables the cloud to decouple the virtual resources from the hardware. It is an open source implementation based on the Xen hypervisor [96]. Eucalyptus internally uses the Xen API to manage the virtual resources. There is also an enterprise version available which works with Vmware ESXi” [89]. Currently it provides for AWS (Amazon Web Services) compatible API so that the web interfaces to manage the resources can provide for interoperability between different cloud infrastructures in the future. Euca2ools is a tool that provides command line interfaces to manage the Eucalyptus resources internally. The CLI is flexible enough to provide for the interaction between the private and various public cloud offerings, for example Amazon EC2.

OpenStack

OpenStack [81] is another IaaS style cloud computing platform initially developed by RackSpace [87] and NASA [88]. "It is an open source implementation, designed to provision and manage large networks of virtual machines, creating a redundant and scalable cloud computing platform. It gives you the software, control panels, and APIs required to orchestrate a cloud, including running instances, managing networks, and controlling access through users and projects [81]". OpenStack Compute can work with different hardware platforms as well as different hypervisors available in the market. More than 150 companies have joined this project and new cloud computing startups are moving towards the OpenStack solutions, to implement their cloud computing services.

VCL (Virtual Computing Lab)

"VCL [86] is a cloud infrastructure developed and deployed in production at NCSU." [82] "VCL is an open-source system used to dynamically provision and broker remote access to a dedicated compute environment for an end-user. The provisioned computers are typically housed in a data center (in the case of NCSU in five data centers) and may be physical blade servers, traditional rack mounted servers, or virtual machines. VCL can also broker access to standalone machines such as lab computers on a university campus [86]". "Resource reservations are through a web interface. VCL can provision undifferentiated bare-metal or virtual machines where user is free to install any software they want and work on it for the time allocated. It can also provide blade servers running high end software (i.e. a CAD, GIS, statistical package or an Enterprise level application) and a cluster of interconnected physical (bare metal) compute nodes" [86]. VCL also has an API that can be used to automate the provisioning. Cost efficiency of VCL and of VCL-HPC is discussed in [91].

Goal of the thesis:

This thesis explores viability, advantages and disadvantages of HPC in the cloud. The environment in which we have assessed the issue is VCL [67]. Different VCL provisioning options, such as preconfigured HPC clusters, a preconfigured GPGPU cluster, and a user defined customized cluster are explored. VCL-HPC uses LSF scheduling, GPGPU host

nodes use Rocks (Rocks is shipped with preinstalled Torque resource manager and Maui scheduler), and in the customized cluster we used Torque resource manager [68] with Maui scheduler [69] to provide VCL resources as compute nodes to the user. An aggregated cluster with certain number of preconfigured nodes can also be provisioned to the user so that the user is free to install customized resource manager and a scheduler to execute their tests in an environment that suits them. Since MPI, OpenMP and GPGPU programming using CUDA is a SIMD (Single Instruction Multiple Data) approach, users and programmers use these libraries to write applications that take advantage of data parallelism. Such applications are tightly coupled. This work explores how the performance of such real time compute intensive applications, varies when executed on different clusters. This thesis details the performance characteristics of HPL (High Performance Linpack) and Ping-Pong (network latency and bandwidth) benchmarks when run on different clusters mentioned above. Further, we discuss new methods of accessing these in-house (VCL-HPC and GPGPU) clusters. We discuss the design and implementation of a gateway server which provides an abstraction in VCL through which these in-house clusters can be accessed via VCL. We discuss the advantages and use cases of this abstraction which will enable efficient and secure implementation of HPC in future cloud infrastructures. We discuss integration of VCL and BlueGene/P as the future of HPC in cloud and explore the possibility of constructing a multi-GPU environment in a network.

Thesis Overview

In VCL we distinguish general purpose resources (VCL-GP) and high performance resources (VCL-HPC). Users can construct clusters in VCL and can access HPC facilities in the following ways.

- Existing clusters in VCL
    - Preconfigured HPC facilities (VCL-HPC)

      This encompasses preconfigured high performance clusters and queues in VCL like the Henry2 and Sam cluster. These clusters are configured with high end machines and high performance network topology. We discuss the topology and

architecture of VCL-HPC in chapter 2. Chapter 3 discusses the performance characteristics of the VCL-HPC (Henry2) cluster.

- Requesting individual reservations

  Users can request individual reservation for separate machines and construct a cluster using any cluster software of their choice. Depending on the privileges users may not have control over the topology or the quality of machines provisioned. Chapter 2 details the steps to construct an individual cluster using Torque resource manager and a Maui scheduler. Performance analysis of this constructed cluster is discussed in chapter 3. Use of Rocks software to construct a cluster over these individual resources is explored in chapter 4.

- Using VCL cluster reservation function

  VCL provides for requesting aggregated resources with certain preconfigured properties as a cluster. The cluster then needs additional software for management purposes. We have not discussed this topic in the thesis.

- Extra clusters

  HPC resources can be requested from external clusters like the NCSU ARC cluster which is not a part of the VCL subsystem. We discuss the topology and architecture of the ARC cluster in chapter 2. Chapter 3 discusses the performance characteristics of the ARC cluster. Access to this preconfigured high performance cluster in VCL is enabled by a gateway (redirection) server. The gateway server is discussed in chapter 4.

- Future Hybrid HPC/HPD facilities

  - While gateway access to external HPC facilities is very useful, it will be really advantageous to have a much finer granularity control over external high performance computing resources (such as individual nodes and cores).

  - Multi-GPU in a network

    GPU's are being widely used for computations but in a single node environment. We discuss the current research in this area and discuss the possibility of multi-

GPU abstraction in a network and ways to access those facilities in VCL. This is discussed in chapter 5.

- VCL integration with BlueGene/P

  We discuss the possibility of VCL communicating with the BlueGene/P resource manager. In the future, VCL would be able to provision clusters running on the BG/P architecture. The design for enabling this service is discussed in chapter 5.

We conclude the discussion in chapter 6. References are provided in following section and Appendices provide a few traces and other details.

Chapter 2

NCSU HPC Resources

There are four types of HPC resources available to NCSU researchers: 1) centrally managed VCL affiliated resources, 2) specialized HPC resources open to NCSU community (such as ARC GPGPU cluster), 3) private (research specific) HPC facilities and clusters, and 4) external resources (e.g., ORNL supercomputers). In this thesis, we limit our interest to HPC-VCL, ARC and general purpose VCL facilities that allow on-demand construction of fixed size or dynamic clusters.

We now describe the VCL-HPC architecture (pre-configured, and available to users for batch HPC processing), the architecture of the ARC GPGPU cluster, and the architecture of VCL on-demand clusters. One thing to note about pre-configured VCL-HPC and other pre-configured NCSU HPC facilities and clusters is that the external access to those facilities (from public IP numbers) is channeled through login node(s), while the computational resources are shielded from the outside and reside on a private network. In general such resources have a tightly coupled (low latency, high inter-node communication intensity) interconnect topology and access to a large amount of either directly attached or network attached storage. On the other hand, on-demand clusters may have a more loosely coupled topology and when used as HPC framework, may be suitable only for applications that required low intensity inter-node communications.

2.1 VCL-HPC

The VCL-HPC [2] services uses one or more login nodes, runs Linux, LSF [98] scheduler, and typically operates about 1000 IBM BladeCenter blades (recent models) in bare-metal mode. The actual size of VCL-HPC varies depending on the demand on the VCL resources – if more resources are needed in the general purpose mode – VCL-GP (e.g., desktops, class or lab reservations or standalone servers), then some of those resources are pulled (logically)

out of VCL-HPC. If there is a surfeit of resources in VCL-GP (e.g., during holidays), those resources are shifted to VCL-HPC. VCL-HPC generates about 13+ million CPU hours per year. VCL-GP results in about 250,000 resource reservations per year.

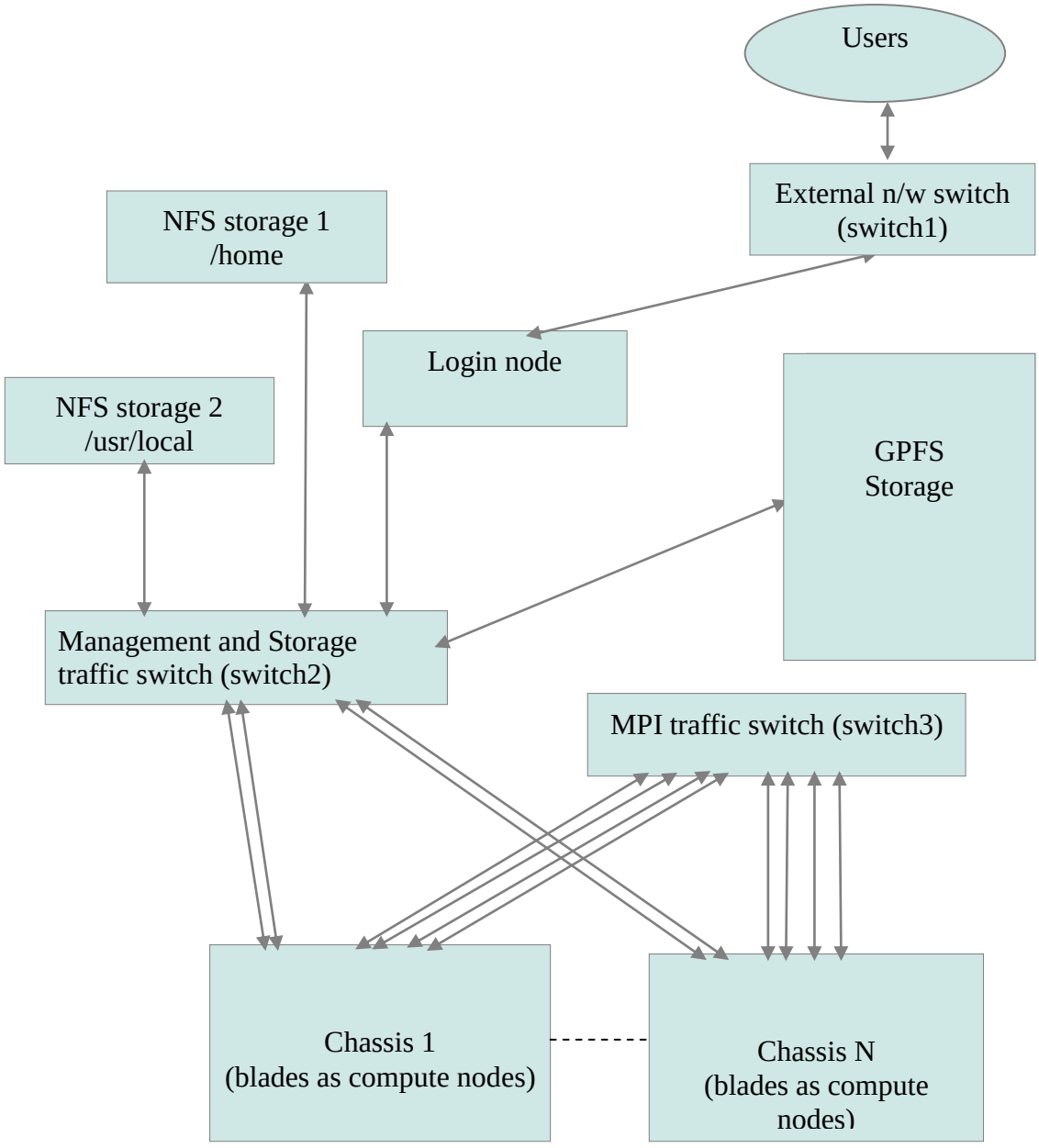

Figure 1: VCL-HPC (Henry2 cluster)

VCL-HPC provides access to two clusters, the Henry2 Cluster and the Sam cluster [2]. We will discuss the Henry2 cluster since we have used the LSF queues configured on the Henry2 cluster for our experimentation. The architecture of the Henry2 cluster is shown in Figure 1. There is a principal login node, and possibly host of on-demand secondary login nodes used for real-time monitoring of specific runs. Login nodes use NFS attached files, and GPFS files to provide data storage. Each computational node has two communication interfaces – an MPI message exchange interface, and a back-end management interface. Typically node interconnects are 1Gbps capable, but more recently 10Gbps sub-clusters have been added. Also there are InfiniBand capable VCL-HPC sub-clusters. Figure 1 illustrates the VCL-HPC architecture.

i. Login nodes:

These are the nodes the users log into from the outside, compile their code on, and invoke LSF scheduler from (submit jobs to Henry2 queues) [98]. A login node has two network interfaces, one interface connects to the external world via the switch1 in Figure 1 and the second interface, which is connected to switch2, provides access the compute nodes and storage. It has access to MVAPICH, MPI, BLAS, etc libraries, as well as Intel (icc, icpc, ifort), GNU (cc, f77) and PGI (pgcc, pgf77, pgf90) compilers, and to other HPC tools. Different queues are configured using LSF to provide mapping among user groups and hardware groups.

ii. Network:

“Henry2 cluster is assembled using two core Ethernet switches - one dedicated for message passing traffic and the second for other network connections needed in the cluster. Each chassis is connected to both networks and contains up to 14 computational nodes. A typical compute node chassis (an IBM BladeCenter chassis uses two aggregated Gigabit Ethernet links for the private HPC network connecting computed nodes to login nodes and storage, and four aggregated Gigabit Ethernet links for the message passing network. The cluster message passing switch can provide up to 384 Gigabit Ethernet ports and can support up to 96 chassis aggregating four links per chassis. Currently Henry2 network architecture will

support about 1344 nodes (this is an approximate number because some of the chassis have separate low latency networks and have a single Gigabit Ethernet link to the core message passing switch). Communications between compute nodes may be non-uniform due to Henry2's network architecture. Nodes within a single chassis can communicate with no bandwidth restriction (full gigabit in each direction) via the chassis backplane. Communications between blades in different chasses is limited to 4 gigabits per second in each direction. So for example if eight nodes in one chassis were exchanging messages with eight nodes in another chassis they could each potentially have a gigabit per second of traffic in each direction for total of 8 gigabits of data per second in each direction. However the Henry2 network architecture would limit the achievable communication rate to four gigabits per second in each direction. The communication could also be impacted by message passing network traffic from the other blades in each chassis that share the aggregated links. In addition to the bandwidth effects of the network design there are also latency effects. Within a chassis messages have a single switch to traverse. For communications between chasses there are three switches that must be traversed (the chassis switch in each chassis plus the core switch). Each network switch adds some additional time before a message reaches its destination [99]".

iii. Storage:

"There are three types of file systems on the Henry2 cluster. However, only two types are generally accessed by users: Network File Systems (NFS) and General Parallel File Systems (GPFS). Directories such as /home, /usr/local, and /share are NFS mounted file systems, While run-time user data support storage such as /gpfs_share is a GPFS file system. File systems on Henry2 use a variety of disk arrays that are typically fibre channel attached to file servers. In general blade servers are used for file servers on Henry2. File server blades are located in chassis with an additional IO module that provides fiber channel connections to each blade server. NFS file systems on Henry2 typically have a single file server with fiber channel connection to a disk array. NFS tends to perform poorly when accessed from a large number of nodes concurrently. GPFS provides capabilities to support concurrent parallel access of a single file from multiple nodes. But, GPFS tends to not perform well for

accessing large numbers of small files. GPFS as configured on Henry2 uses four servers that create network shared disks (NSDs). These NSDs create a file system that is mounted on each Henry2 login and compute node. GPFS input/output (I/O) operations are cached on the local node and then synchronized with the file system. Multiple NSD servers are able to support more concurrent use and also provide resiliency against failures [99]".

iv. Nodes:

"Computational nodes are typically IBM BladeServer blades. Each node runs CentOS release 5.4, and has access to appropriate math, MVAPICH, and MPI libraries and various compilers. Henry2 nodes are typically dual Xeon blade servers. But, there are a mix of single-, dual-, and quad-core Xeon processors. Nodes typically have 2-3GB of memory per processor core and a modest size disk drive that is used to hold the operating system, swap space, and a small local scratch space. Nodes have two Gigabit Ethernet interfaces (10 Gbps in some chasses). In compute nodes one of these interfaces is used for a private network connecting the compute nodes to the login nodes and to cluster storage resources. The second Gigabit Ethernet interface is used for a private network dedicated to message passing traffic. On login nodes one interface is used for access from the Internet and the second for access to compute nodes and storage [99]".

2.2 ARC: A Root Cluster for Research into Scalable Computer Systems

ARC [3] is a specialized cluster of 1U SuperMicro GPGPU nodes with 1 GPU per node. There are 108 such nodes and a separate login node with similar configuration. Rocks 5.3/CentOS 5.7 Linux x86_64 is the base operating system on the ARC nodes. Rocks, is a cluster management suite which ships with preinstalled Torque/OpenPBS and the Maui scheduler packages for resource management and job submission. More information can be found at [3]

ARC cluster architecture:

The internal architecture of the GPGPU ARC cluster at NC State is as shown in Figure 2. It consists of a head node, a login node, network switch and 108 node cluster of 1U SuperMicro nodes with one nVidia GPU per node. The nodes have one Mellanox InfiniBand adapter card each (configured at 40Gb/s). They are connected to each other via Mellanox InfiniScale switches Description and purpose of each component follows:

i. Head Node:

Head node is the main interface to the outside world and the users. The users login to the head node and are automatically redirected to the cluster login node. The PBS server daemon runs on the head node. The head node (pbs server) is responsible for scheduling, queuing and forwarding of the jobs to the computational nodes in the GPGPU cluster. Jobs are submitted via the PBS (Maui scheduler). As soon as the job is queued, the head node (using Maui scheduler) looks for the first unused GPU node in the cluster (or for the total number of nodes requested by the user) and sends the job over to the node to be executed. The granularity of access is a node and not a single core as is the case with the LSF scheduler in the Henry2 cluster. The scheduler controls the job queuing time and has control over the execution time. If a job takes more than the allocated time the job is killed and the node is freed to execute new jobs.

ii. Firewall:

The entire cluster is managed by Rocks [5] and is firewalled from the outside. Rocks, requires the cluster to be on a private network. Rocks, discovers computational nodes over Ethernet and installs Linux on the local hard drive of these nodes over PXE. The head node is outside the firewall to allow users to get to the login node. One needs to be in the college network to access the ARC cluster.

iii. Login Node:

The login nodes are used to compile and proof run the user code. The login node has the CUDA, MPI, MVAPICH, BLAS, and other libraries installed. GNU (gcc) and PGI (pgcc,

pgf77, pgf95, pgCC) compilers and other tools are installed. The user is responsible for any other required libraries if those are required. The login node is the same in hardware configuration (see Table 2) as the 108 other cluster compute nodes, except that it does not take part in any compute activities. PBS client is not installed on this machine and hence the PBS server cannot send jobs to this machine. Login node has a GPU unit attached to it so as to enable users to compile CUDA based code. Once the code is compiled on the login node, it is submitted to the scheduler queue using the PBS (Maui) commands (qsub command). Details on how to get started, compile the code and submit jobs to the scheduler can be found on the website [3].

iv. NFS Attached Storage

ARC users have all their data on the login server assigned to them. Data is made available to computational nodes via NFS.

v. Computational nodes:

Each node has a GPGPU unit (nVidia GPU card) as well as the runtime libraries required to execute user applications. The Maui scheduler (pbs_server) running on the head node selects one of these nodes (or the number of nodes requested by the user) to schedule the job.. Each node has a 120 GB SSD drive to boost the performance even further. Some nodes differ in the type of GPU card installed on them. The various GPU cards available are nVidia C2070, nVidia C2050 and nVidia GTX480 [3]. Access to different GPU card groups is controlled through scheduler queues.

The system is monitored using Ganglia [101]. It shows the total number of CPU units, GPU units, hosts that are operational, number of hosts that are down, average load on the cluster, memory utilization, job queue, job status, etc

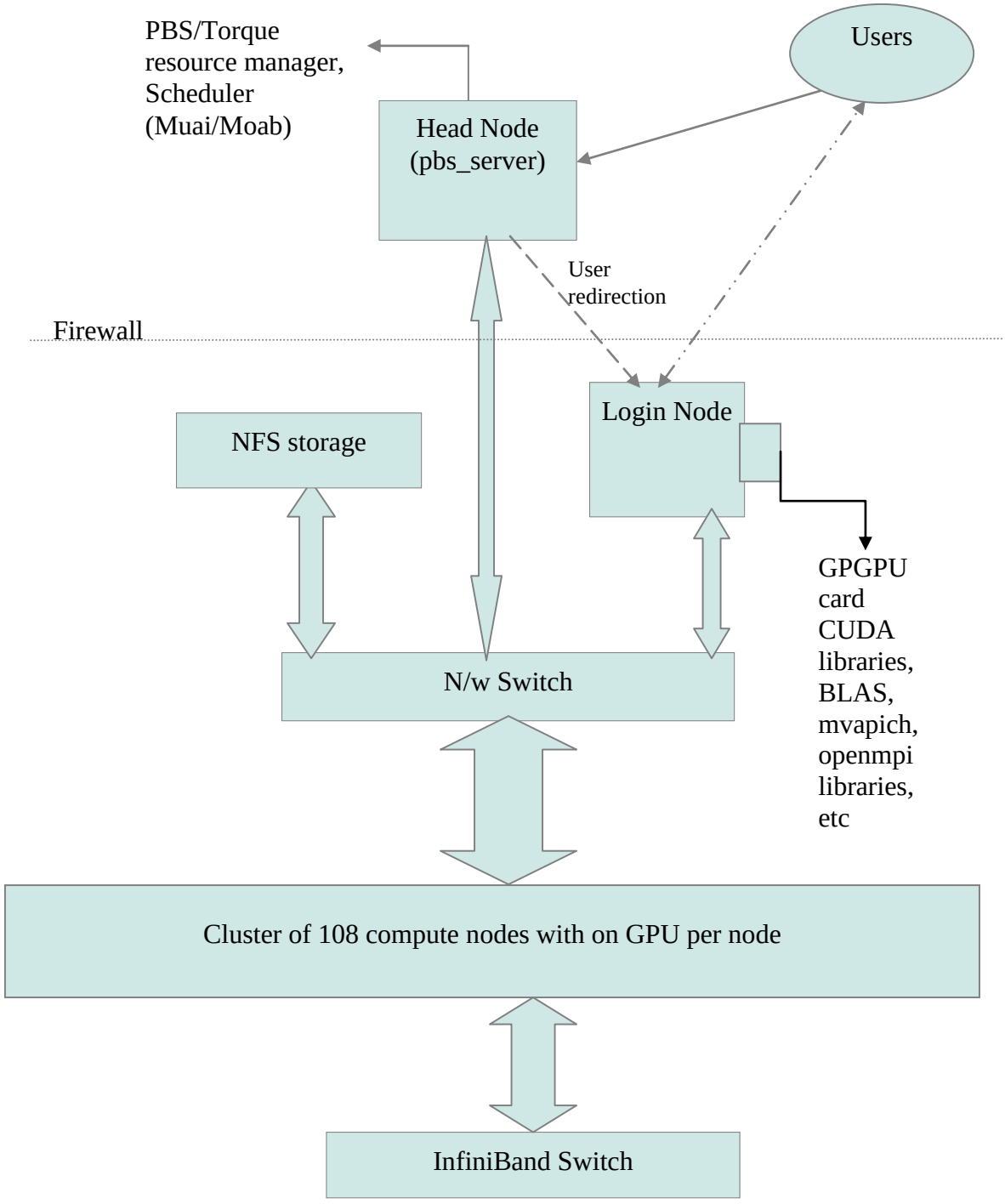

Figure 2: Arc Cluster

2.2 General Purpose User Defined Clusters in VCL

We also have designed a user defined cluster using VCL general purposes resources. These resources are provisioned by the VCL management node and a user may not have control over the configuration topology or computational node profiles beyond general infrastructure characteristics such as the type of operating system on the node and CPU speed.

2.2.1 VCL Compute Cluster using Torque resource manager and Maui scheduler

Figure 3 illustrates the overall view of a general user constructed cluster in VCL (in this case comprised of 5 nodes). The user can request resources from VCL, via the user web interface, either individually (one node at the time) or using cluster reservation option. In either case, these resources are normal VCL resources, such as Linux based machines (CentOS 5.4), except that they may be logically structured as shown in Figure 3. Depending on the user privileges, it may be possible to request the resources to be within a particular rack or chassis and to have particular characteristics. However, in most cases, the resources may come from anywhere in the VCL cloud (even different data centers). This will impact physical topology of the network interconnects and homogeneity of the provisioned computational resources.

There is a head node which runs whatever the user installs on it – for example in our example case, Torque (pbs_server) resource manager [68] with a Maui scheduler [69] (pbs_mom) on Linux. The other nodes run as worker nodes or basic compute nodes. The head node, which also acts as a login node, distributes jobs to the compute nodes. All the nodes have the OpenMPI configured

As an example, simple jobs submission to pbs_server would look like this

```
$>  echo “Nikhil” | qsub
```

A certain number of compute nodes can be requested using the command:

```
$> qsub –I -q batch -l nodes=4:ppn=1
```

This starts a job on four nodes and provides the user with an interactive shell in one of the nodes. Jobs can now be launched using the command:

```
$> mpirun -np 4 ./executible
```

More complex jobs can be submitted by writing a PBS script. More information can be found on the Torque website [68].

Following steps need to be followed to setup a general purpose cluster (five node cluster):

- Request five reservations for similar machines from VCL
- Setup ssh keys between the VCL nodes so that they can talk to each other securely and without using passwords.
- Setup ssh keys for root as well as the user who is going to run the jobs in the end.
- Setup a DNS service of some kind for the nodes. This can be done by either leveraging normal VCL DNS services and/or by adding entries to the /etc/hosts file. We have used the /etc/hosts file method.
- Setup a shared storage between these servers via NFS or any NAS solution available.
- Open the firewall ports that Torque and Maui use.
- Download and install Torque or PBS on head node (pbs_server)
- Configure the head node and the compute nodes in Torque configuration files
- Install the PBS compute node packages on the slave nodes (pbs_mom)
- Download and install the Maui scheduler service on the head node
- Start the PBS and the Maui services on the Head node as well as the compute nodes.
- At this point the compute nodes should be seen as “pbsnodes” on the Head node.
- Install the Torque aware OpenMPI library on the Head node.
- Share this library over the shared storage and change library and default paths to point to this shared storage.
- Now, the cluster is ready to execute MPI jobs

The HPCC benchmarks [105] (HPL and Ping Pong) were run on this cluster. The next chapter deals with the performance characteristics of this simple high compute cluster based on VCL general purpose resources.

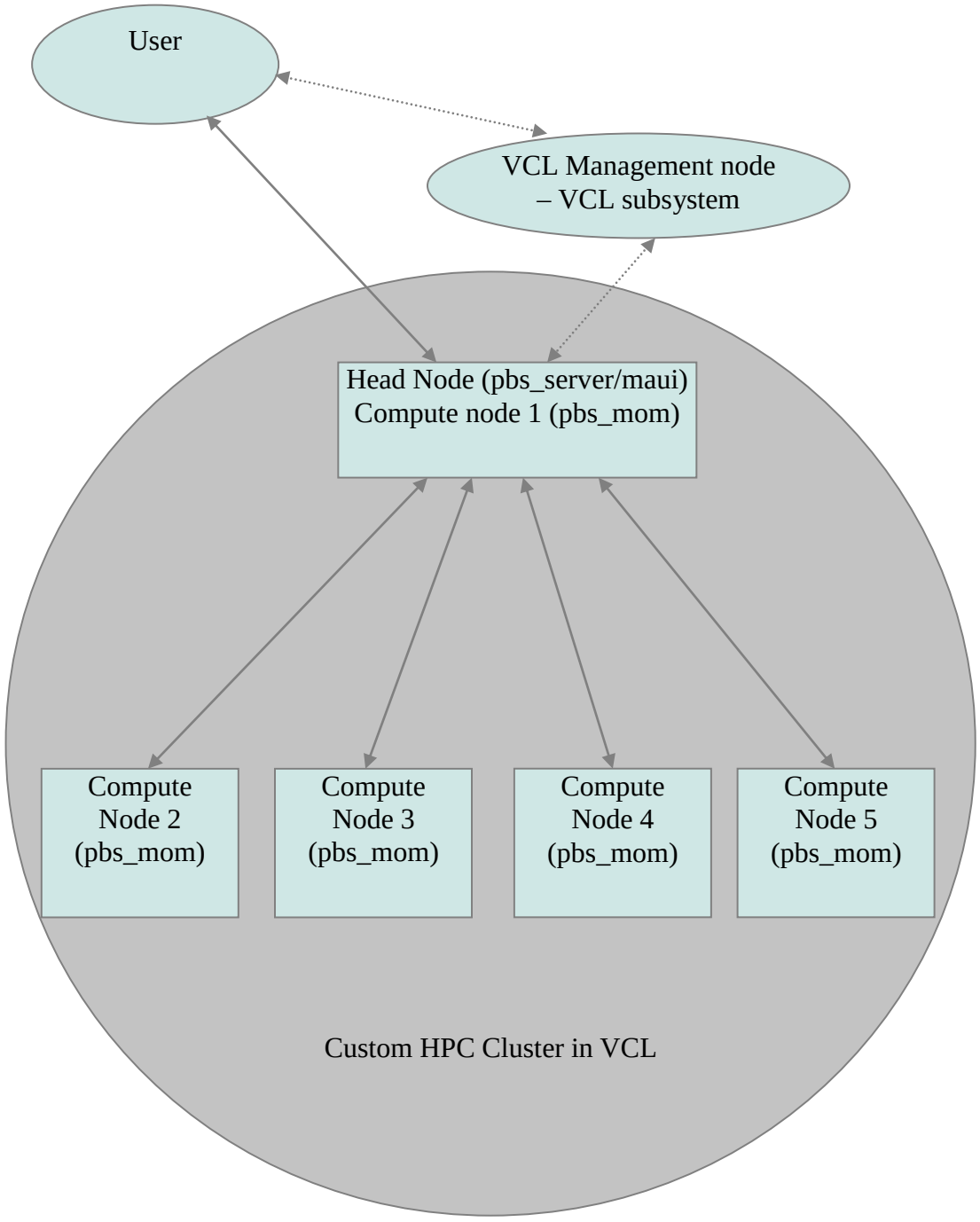

Figure 3: HPC in VCL (VCL-GP)

2.3 A discussion of Hadoop cluster on VCL nodes

Depending on the interconnect topology and nodes, one may be able to use the on-demand cluster as a more tightly coupled HPC cluster, or perhaps as an High Performance Data (HPD) cluster, or a general loosely coupled computational engine., HPD is concerned with intelligent data distribution so that applications can take advantage of data parallelism. Map reduce techniques have been instrumental in achieving good performance for applications working on highly parallel and distributed data. The essence hinges on high parallelism for large but almost independent data sets (such as those found in the context of unstructured data sets/files that need to be independently searched) and on relatively low intensity low latency sensitivity inter-node communication traffic levels Apache Hadoop [71] is an open source implementation of the Map-Reduce approach. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model [71]. The Hadoop library can work on jobs with data sets on a single machine to thousands of machines. Hadoop provides various features like high availability, fail-over techniques, load balancing, etc. The HDFS (Hadoop Distributed File System) distributes the application data to achieve the required scalability. Since the application data placement is handled by the HDFS layer, it has the capability to distribute and locate the application data as requested thus the paradigm of Map Reduce can now be applied to the application data using a simple programming model.

We can map Hadoop into a distributed heterogeneous environment such as what can be provided through a general purpose VCL resource provisioning. Installation of Apache Hadoop is straightforward and well defined. The instructions can be found at [71]. Once Hadoop is installed, the jobs can be mapped to the cluster and VCL can thus provide a Hadoop cluster. A number of Hadoop experiments have already been performed on VCL and there are working Hadoop images and clusters on VCL. However, performance evaluation of these loosely coupled Hadoop clusters is not part of this study.

It is worth noting, thought, that the main reason for performance degradation of HPC clusters is often the latency introduced by the network interconnects. This is evident in the ping-pong latency test shown in chapter 3. In a Hadoop environment there is minimum communications among threads. Different worker threads affect different parts of the application data. This data and worker threads are mapped by Hadoop in such a way that the data, a worker threads works on, is located locally on that node on which the worker thread runs. This reduces network latency. The reduction threads reduce the processed data which is again local to those threads. This improves the overall system performance. Hence, a Hadoop based implementation can perform well in a VCL-GP environment even in situations where a reservation may results in higher-latency inter-node communications.

Chapter 3

Performance

3.1 The LINPACK Benchmark

The LINPACK Benchmark was introduced by Jack Dongarra et al. in 1979 [26]. It is available from at [10]. “The Linpack Benchmark is something that grew out of the Linpack software project. It was originally intended to give users of the package a feeling for how long it would take to solve certain numerical problems. The benchmark stated as an appendix to the Linpack Users' Guide and has grown considerably since the first Linpack User’s Guide was published in 1979 [55]”.

“The Linpack Benchmark is a measure of a computer’s floating-point abilities and speed. It is determined by running a computer program that solves a dense system of linear equations [55]. The LINPACK benchmark is used as a measure of performance by the Top500 [44] supercomputer initiative. By measuring the actual performance for different problem sizes N, a user can get a range of useful data – including, for example, maximal achieved performance Rmax for the problem size Nmax, the problem size N1/2 where half of the performance Rmax is achieved, etc.. These numbers together with the theoretical peak performance Rpeak are the numbers given in the TOP500 surveys. In an attempt to obtain uniformity in performance reporting, the algorithm used in solving the system of equations in the benchmark procedure must conform to LU factorization with partial pivoting. In particular, the operation count for the algorithm must be 2/3 n^3 + O(n^2) double precision floating point operations. This excludes the use of a fast matrix multiply algorithm like "Strassen's Method" or algorithms which compute a solution in a precision lower than full precision (e.g., 64 bit floating point arithmetic) and refine the solution using an iterative approach [57]”.

LINPACK measures the performance of an HPC cluster by calculating the number of floating point operations per second or FLOPS the cluster can perform. The performance is generally reported in millions of floating point operations per second or gigaflops per second. The Top500 [44] community reports the performance in MFLOPS or mega floating point operations per second.

3.1.1 High Performance LINPACK (HPL)

High Performance Linpack (HPL) [10] is the parallel and unrestricted version (size of coefficient matrix is flexible and can be changed manually) of the LINPACK benchmark (LINPACK100 [55], LINPACK1000 [85]) which solves a random dense linear system of equations. HPL performs double precision (64 bits, 8 bytes) arithmetic on distributed-memory computers. HPL is a portable open source implementation of the High Performance Computing Linpack Benchmark [10].

The algorithm used by HPL can be summarized by the following keywords: "Two-dimensional block-cyclic data distribution - Right-looking variant of the LU factorization with row partial pivoting featuring multiple look-ahead depths - Recursive panel factorization with pivot search and column broadcast combined - Various virtual panel broadcast topologies - bandwidth reducing swap-broadcast algorithm - backward substitution with look-ahead of depth one" [10].

The output of HPL [10] provides values for accuracy, detailed timing, the total computation time required and the performance in GFLOPS (Floating point operations per second). HPL requires Message Passing Interface MPI, Basic Linear Algebra Subprograms BLAS or Vector Signal Image Processing Library VSIPL libraries installed on the system [10].

3.1.1.1 HPL algorithm

- "HPL solves a dense linear system of order n: A x = b by first computing the LU factorization with row partial pivoting of the n-by-n+1 coefficient matrix [A b] =

[[L,U] y]. Since the lower triangular factor L is applied to b as the factorization progresses, the solution x is obtained by solving the upper triangular system U x = y. The lower triangular matrix L is left unpivoted and the array of pivots is not returned" [10].

- "The data is distributed onto a two-dimensional P-by-Q grid of processes according to the block-cyclic scheme to ensure "good" load balance as well as the scalability of the algorithm. The n-by-n+1 coefficient matrix is first logically partitioned into nb-by-nb blocks, that are cyclically "dealt" onto the P-by-Q process grid. This is done in both dimensions of the matrix" [10].
- "The right-looking variant has been chosen for the main loop of the LU factorization. This means that at each iteration of the loop a panel of nb columns is factorized, and the trailing submatrix is updated. Note that this computation is thus logically partitioned with the same block size nb that was used for the data distribution" [10].

3.1.1.2 HPL performance tuning parameters

The various tuning parameters are available in the HPL.dat file. Following are some of the parameters:

- N's: This is the problem size. These values indicate the order of the coefficient matrix. The bigger the matrix size, more the floating point operations. For better performance, the matrix needs to fit in the physical memory of the machine [10].
- NB: NB is the block size. It is basically the block size used during the LU block factorization of the matrix. A parallel LU algorithm calculates the LU factorization with block granularity and this factor determines the block size for factorization [10].
- P,Q: P: Rows in process grid. Q: Columns in process grid

  P,Q indicate the distribution of the blocks (blocksize is determined by NB) to the process grid. If either P, Q or NB is equal to one we will have vector-matrix operations. If NB is one, we will always have vector-vector operations irrespective of P and Q. If NB is not one, P and Q are not one then we will have matrix-matrix operations [10].

Parameters not considered in this scenario:

- “Depth: This factor specifies the look-ahead depth used by HPL. A depth of 0 means that the next panel is factorized after the update by the current panel is completely finished. A depth of 1 means that the next panel is immediately factorized after being updated. The update by the current panel is then finished [10].
- BCASTS: In the main loop of the algorithm, the current panel of column is broadcast in process rows using a virtual ring topology. HPL offers various choices and one most likely want to use the increasing ring modified encoded as 1. 3 and 4 are also good choices [10].
- RFACTS, PFACTS, NDIV, NBMIN: These parameters specify the algorithmic features. HPL will run all possible combinations of those for each problem size, block size, process grid combination. This is handy when one looks for an "optimal" set of parameters. To understand a little bit better, let say first a few words about the algorithm implemented in HPL. Basically this is a right-looking version with row-partial pivoting. The panel factorization is matrix-matrix operation based and recursive, dividing the panel into NDIV subpanels at each step. This part of the panel factorization is denoted below by "recursive panel fact. (RFACT)". The recursion stops when the current panel is made of less than or equal to NBMIN columns. At that point, xhpl uses a matrix-vector operation based factorization denoted below by "PFACTs". Classic recursion would then use NDIV=2, NBMIN=1. There are essentially 3 numerically equivalent LU factorization algorithm variants (left-looking, Crout and right-looking). In HPL, one can choose every one of those for the RFACT, as well as the PFACT [10].
- Threshold: This parameter specifies the threshold to which the residuals should be compared with. The residuals should be or order 1, but are in practice slightly less than this, typically 0.001. This line is made of a real number, the rest is not significant. For example: “16.0 threshold”. In practice, a value of 16.0 will cover most cases. For various reasons, it is possible that some of the residuals become slightly larger, say for example 35.6. HPL will flag those runs as failed, however they can be considered as correct. A run should be considered as failed if the residual is a

few order of magnitude bigger than 1 for example 10^6 or more. Note: if one was to specify a threshold of 0.0, all tests would be flagged as failed, even though the answer is likely to be correct. It is allowed to specify a negative value for this threshold, in which case the checks will be by-passed, no matter what the threshold value is, as soon as it is negative. This feature allows to save time when performing a lot of experiments, say for instance during the tuning phase. Example: -16.0 threshold [10].

- Swap: This parameter allows specifying the swapping algorithm used by HPL for all tests. There are currently two swapping algorithms available, one based on "binary exchange" and the other one based on a "spread-roll" procedure (also called "long" below). For large problem sizes, this last one is likely to be more efficient [10].

- L1,U: This parameter specifies whether the upper triangle or the panel of rows is stored in transpose for or not [10].
- Align: This parameter allows specifying the alignment in memory for the memory space allocated by HPL. On modern machines, one probably wants to use 4, 8 or 16. This may result in a tiny amount of memory wasted [10]".

HPL was executed on the clusters mentioned in the previous chapter and the performance of HPL on different clusters is presented in the following sections. Cluster configurations are identified below along with the results.

Some HPL runs were done on single core processors, and some were done on multi core processors. Single core runs for sample configurations are given in Appendix B as is a possible theoretical limit on the performance of a processor core. The processor performance depends on various factors which are very much related to the microprocessor architecture and the test being run. Asim Nisar et al. in [94] give a deeper insight on the factors that affect performance for a 45nm Core2 processor.

Another interesting fact to be noted regarding the experiments is that when we work on a specific CPU configuration, it means that we have requested the resources which have

similar architecture and network interconnects. However, the resource manager maintains different job queues for different hardware/platforms. The resource manager does not guarantee that the job will run on the same machine every time. The job can run on different machines (e.g: different core architectures) at different times but belonging to the same queue and having the same or very similar hardware and software configuration.

3.2 Performance evaluation of compute jobs on VCL-GP nodes

A possible VCL node configuration is shown in Table 1 below. All the results were taken on nodes with similar configuration. Various tests were run in order to characterize components of this cluster, and composites of these components, with regards to performance. This includes CPU utilization stats.

Table 1: CPU Configuration 1: A typical node in VCL-GP cluster

Cluster	VCL cluster
Queue	Batch
Processor Model	Xeon L5638
Microprocessor Architecture	Westmere-EP
Clock Frequency	2 GHz
Number of Processors	1
Number of Cores	1
Hyper threading	NO
L1 cache	32k-Instr, 32k data per core
L2 cache	256K-Unified per core
L3 cache	12288K-Unified per processor
Memory	1 GBytes
Network	1 G Ethernet
Operating System	CentOS release 5.4 (Final)
OS architecture	X_86, i686

I. HPL Single node test:

The single node test gives an insight into the actual processor performance on a single node in a cluster. Figure 4 details the result in a graphical format (The horizontal axis represent blocksize and vertical axis represents performance in GFLOPS). The results indicate that a single core performance could reach 7 GFLOPS. This indicates that if we increase the cluster

by adding more nodes we might get better performance. This is explored in the next section where we test the performance of HPL on an on-demand cluster of VCL-GP nodes.

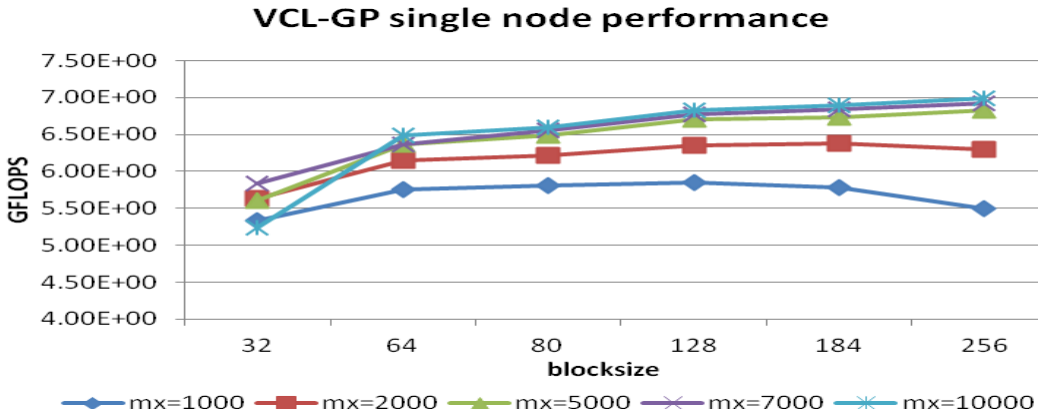

Figure 4: HPL results for varying blocksize on a single VCL-GP node (1 core)

II. Four node VCL-GP Cluster using HPL with varying Matrix size (N):

We constructed a four computational node cluster using VCL-GP nodes (of the same type as in I above). The maximum performance achieved by HPL tests with varying Matrix sizes is shown in Figure 5 (horizontal axis is the matrix size (time at which the test was run) (blocksize at which the maximum performance for this matrix size was observed) and vertical axis show the performance in GFLOPS). Note that in the figure, at all four times the same reservation (same 4 node cluster) exists, but T1 and T2 were nearly 2 days apart (T1 for 10000 order matrix took more than a day to complete). T3 was performed the same day after T2 and T4 was performed a week later T3.

It is interesting to note that measurements taken at different times provided different results. At time T1 measured performance was in the 1 to 2 GFLOPS range, at time T3 (on the same cluster, about 2 days later) achieved 14 to 18 GFLOPS. We speculate that the difference was in large part, due to a possible congestion introduced by other loads in the cluster neighborhood (but not on the cluster machines, they were bare-metal loads). However, some other factors may also have been in the play. More detailed graphs are described in section 3.6 to explain this variability.

As we can see in the Figure 5 that increasing the matrix size tends to increases the performance. As we increase the matrix size, we fill the processor pipeline with more floats. This ensures that there is little or no idle time for the processor. Increasing the Matrix size also makes sure that the processor pipeline has fewer bubbles. Since HPL takes advantage of temporal as well as spatial locality of data (spatial locality is not as highly exploited as temporal locality in HPL), the more data fills up the cache, and the larger is the floating point rate. However, this holds only up to a point. There might be a stage where increasing the matrix size degrades the performance.

For example, with reference to Figure 5, at time T1 the performance increases with the increase in the matrix size. However performance degrades for matrix size of 10000. At time T3, we observed that the performance increases until matrix size 10000, in fact it reaches up to 18 GFLOPS on a four node cluster and an average aggregate performance of about 4.5 GFLOPS per processor. Note that this graph shows the maximum performance from all the tests. The maximum performance was observed at different block sizes for different matrix sizes, at different times as shown in the figure.

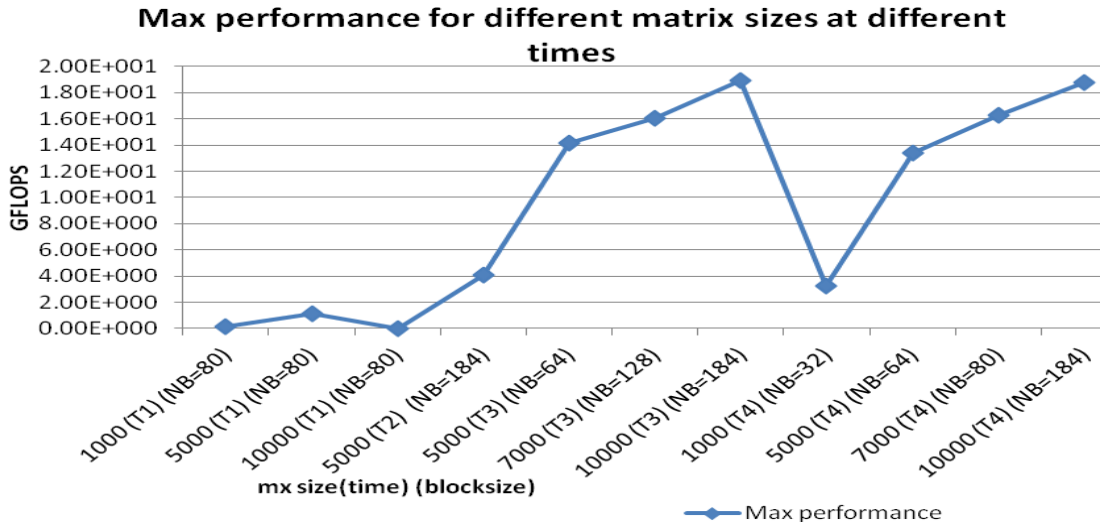

Figure 5: Maximum HPL performance for varying matrix size (N) (blocksize (NB) at which max performance was observed is noted). (Test: 4 nodes, 4 cores). Measurements were taken at different times (T1, T2 → T1+2 days and T3→ T2 + 2 hours)

III. HPL with varying blocksize (NB):

Results of HPL on VCL-GP (4 node cluster) with varying block are shown in Figure 6. The horizontal axis represents blocksize and vertical axis represents performance in GFLOPS. The results measured at different times produced different results due to, for most part, the difference in network bandwidth at different times. We see that increasing the block size increases the performance up to a maximum point. Further increase in the block size causes degradation in performance. It can be argued that as we increase the block size, we increase the advantages of temporal locality. But if we go on increasing the block size, the advantage of data parallelism is lost and hence we see degradation in performance.

It was also observed that larger blocksize can perform better on bigger matrices. This is because in a bigger matrix, there is a larger scope to take advantage of data parallelism. Hence, increasing the blocksize increases the advantages of temporal locality without affecting the data parallelism. The important distinction in this case is that the effect of blocksize followed a similar pattern at two different times T2 and T3. This proves that the HPL algorithm had no glitches but the difference in performance at times T2 and T3 was purely due to configuration related issues (change in network parameters in this case).

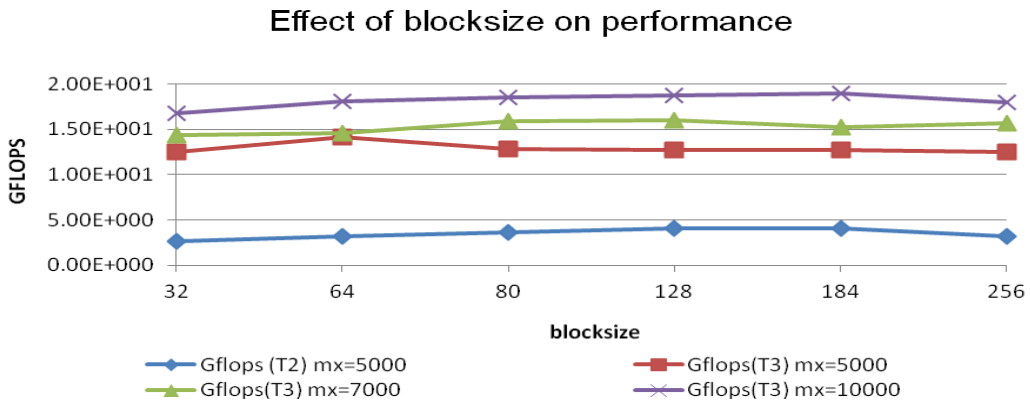

Figure 6: HPL performance with varying block size (NB), (Test: 4 nodes, 4 cores, T2 → T1+2 days and T3→ T2 + 2 hours)

IV. A simple analysis of the Ping-Pong benchmark results:

Ping-pong, a test in the HPCC suite of benchmarks [105], provides a measure of network bandwidth and latency of MPI messages. The latency graph as shown in Figure 7 (horizontal axis represents time (order of matrix used for the test) and the vertical axis represents the Ping Pong MPI latency for 8 byte messages in milliseconds, the latencies map to left hand scale while the performance maps to the right hand scale. The performance always maps to the right hand scale henceforth). It shows that the Ping Pong latency for 8 Byte messages was almost constant at different times and in the 0.06 to 0.09 ms range while the natural order and random order latencies were in the range of 0.05 to 0.07 milliseconds.

However, it is interesting to note that the bandwidth observed at different times was different. As shown in Figure 8 (horizontal axis represents time (order of matrix used for the test) and vertical axis represent the network bandwidth in MegaBytes/second), the bandwidth at times T1, T2 and T3/T4 goes on increasing respectively. This seems to be reflected, in part, in the performance of HPL as shown in Figure 5. The performance at time T3 was the maximum peak performance achieved. Although the latency for 8 Byte messages hardly varied (0.06 to 0.09 ms range), there were changes observed in network bandwidth. Network bandwidth is exploited when large data sets are transferred over the network. In a Ping Pong test the bandwidth is high when 2,000,000 Byte messages are sent across. Similarly, when dealing with larger matrix sizes large data sets are transferred across the network which is why we see a difference in the overall performance.

Since we did not see any difference in the max latency for 8 byte messages at times T1, T, T3 and T4 we investigated further. Deep study considering the latency observed for 2 MB messages was done to understand this behavior and the results are explained in section 3.6. It is important to note for interested readers, that Ping Pong calculates a “stride” for sending the ping and pong in between client and server (with strides greater than 1 nodes are skipped and not all nodes participate in the test) for the latency and bandwidth test based on an initial test run. The results shown for the Ping Pong test throughout the document are the results of the Ping Pong test after the stride has been calculated and applied.

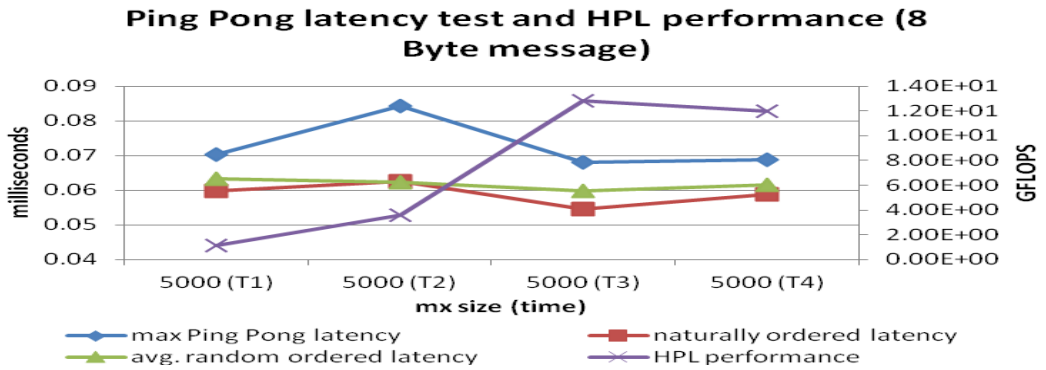

Figure 7: Ping-Pong network (MPI) latency, (Test: 8 Byte message, 4 nodes, 4 cores, T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week)

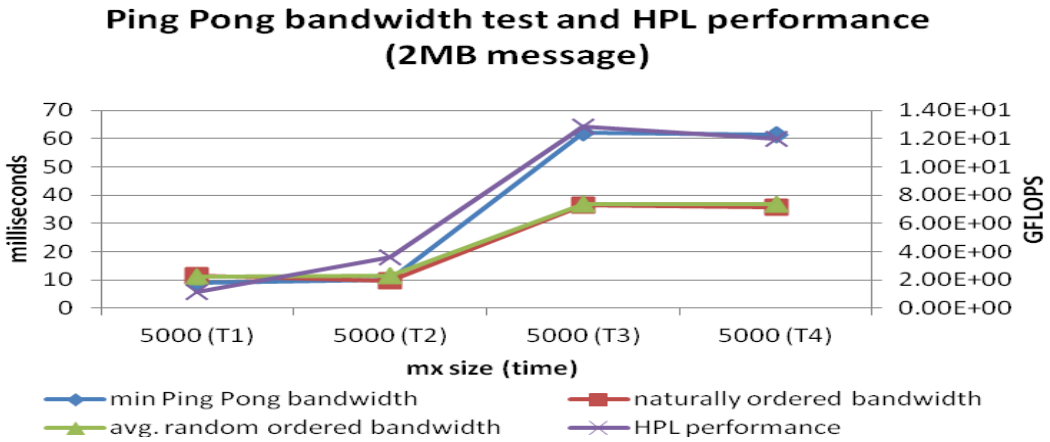

Figure 8: Ping-Pong network (MPI) bandwidth, (Test: 2MB messages, 4 nodes, 4 cores, T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week)

V. HPL performance on single node and multi-node configuration:

We investigated CPU utilization during the HPL tests. The CPU utilization of one of the nodes in the four node cluster is shown in Figure 9 (horizontal axis represents relative time with interval of 1 seconds and vertical axis represent the percentage utilization of CPU). All the other nodes have a similar graph for CPU utilization. Since this particular set of VCL nodes had a single processor with a single core per node, the processor not only had to deal with the floating point computations, but also had to do the network-related and other processing.

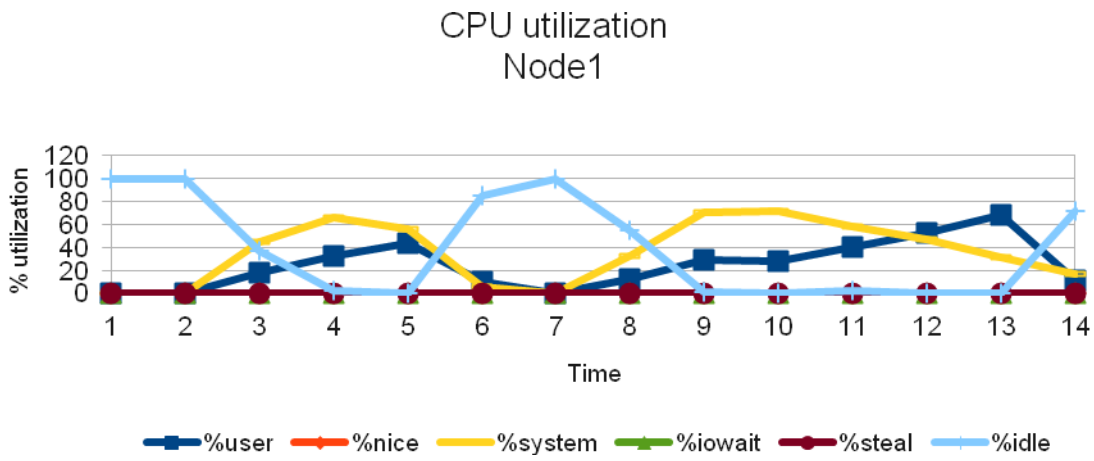

Figure 9: CPU utilization of a VCL node in the cluster, (Test: 4 nodes, 4 cores). Relative time on the x-axis is in seconds.

We see that the neither system CPU utilization (network stack processing) nor the user CPU utilization (floating point computations) exceeds about 60%. The compute processes appears to get less than 60% of the CPU time.

The results of a similar test with HPL being run on a single node with no network packets is shown in Figure 10 (horizontal axis represents relative time with interval of 1 seconds and vertical axis represent the percentage utilization of CPU) depicts that the processor was utilized to its full potential and just for computations. The user process (HPL computations) executes on the CPU with 100% utilization when the test was running. The difference in performance in the above graphs proves that HPC in cloud cannot guarantee performance stability. These observations bring out an important conclusion that HPC in cloud is not perfect.

For the VCL-GP tests the following version of compilers and libraries were used.

Open-MPI version: 1.4.5

Maui resource scheduler: 3.3.1

Torque resource manager: 4.0

Mpicc version: gcc (GCC) 4.1.2

HPL code available from [105]

GotoBlas2 math library: Optimized GotoBLAS2 libraries version 1.13 available from [90]

CC FLAGS: -fomit-frame-pointer -O3 -funroll-loops -Wall

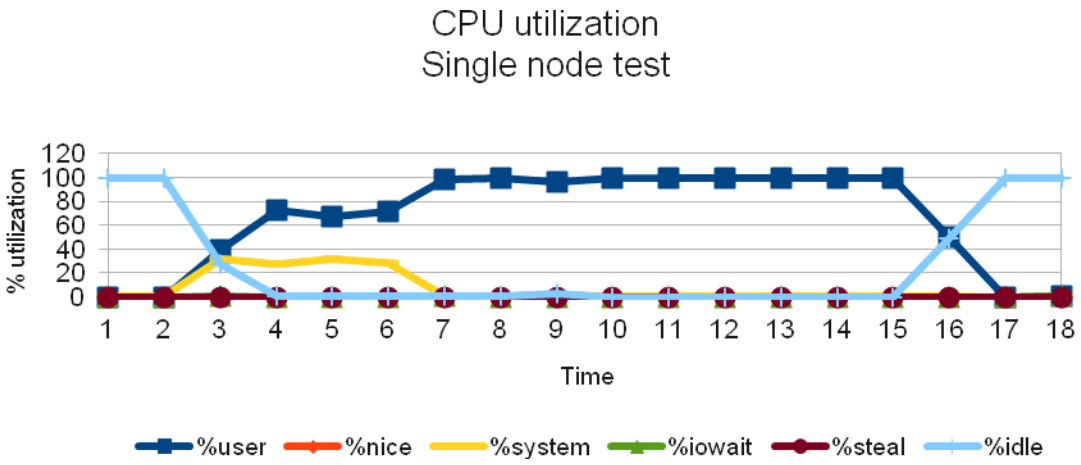

Figure 10: CPU utilization, single node test (1 core)

3.3 Performance evaluation of compute jobs on ARC cluster

ARC cluster was described in Chapter 2. For the ARC cluster tests the following version of compilers and libraries were used.

Open-MPI version: 1.5.4

Rocks: 5.3 (Rolled Tacos)

HPL code available from [105]

HPL-GPU code available from the NVIDIA Developer site [21] (The HPL algorithm for CPU+GPU and implementation is described in the paper [22])

Mpicc version: gcc (GCC) 4.1.2

GotoBlas2 math library: Optimized GotoBLAS2 libraries version 1.13 available from [90]

HPL CC FLAGS: -fomit-frame-pointer -O3 -funroll-loops –Wall

CUDA toolkit version: 4.1 (RC1)

NVCC version: Cuda compilation tools, release 4.1, V0.2.1221

HPL-GPU CC FLAGS: -fomit-frame-pointer -W -Wall -fopenmp –g

Table 2: CPU Configuration 2: A node in the ARC cluster

Cluster	ARC cluster
Queue	c2050
Processor Model	AMD Opetron 6128
Microprocessor Architecture	Magny Cours (K10)
Clock Frequency	2 GHz
Number of Processors	2
Number of Cores	16
Hyper threading	NO
L1 cache	64k-Instr, 64k data per core
L2 cache	512K-Unified per core
L3 cache	5118K-Unified per processor
Memory	32 GB
Network	40Gb/s Infiniband (configurable)
Operating System	CentOS release 5.7 (Final)
OS architecture	X86_64

3.3.1 CPU cluster performance characterization

Following tests were performed to characterize the CPU performance of the ARC cluster.

I. HPL performance on a single node in ARC cluster:

A single node test on ARC cluster was done to ensure that the CPU utilization is good enough for further results. A single node test as shown in Figure 11 (horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS) depicts that we get around 102 GFLOPS per sixteen processors/cores i.e. 6.3 GFLOPS per core.

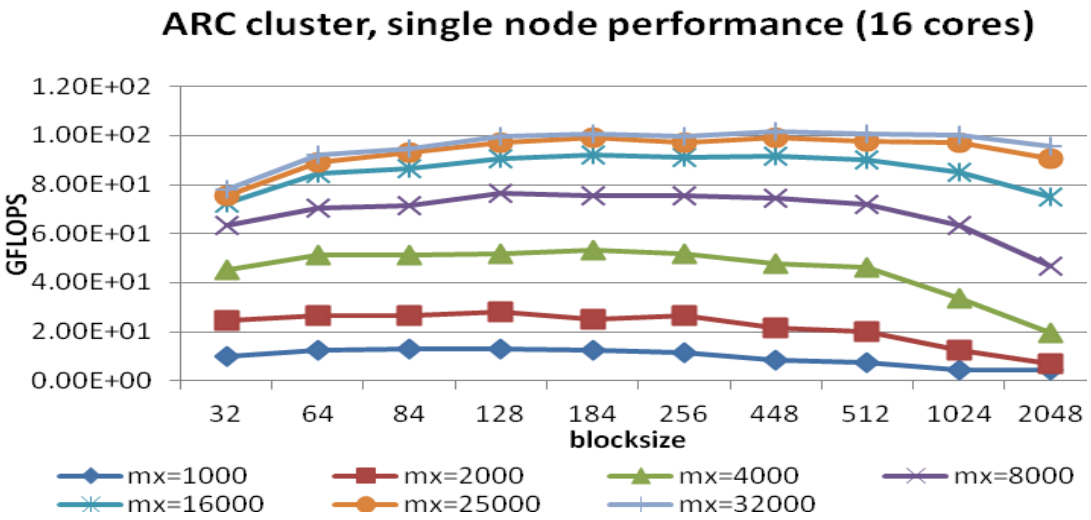

Figure 11: HPL Performance, 1 node (16 cores)

II. HPL with varying Matrix size and blocksize (N, NB):

Figure 12 (horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS), illustrates that as we go on increasing the Matrix size the performance increases. This is because of the linear scaling of the HPL algorithm. With increasing block sizes, we see that a point of maximum performance is reached, and then the performance drops. The point of maximum performance is different for different Matrix sizes. For a 64,000 ordered matrix, the performance starts degrading for blocksize 84 and keeps dropping for larger blocks. For order 32,000 matrix, the performance drops after blocksize of 184. The graph does not give a clear single point of maximum for all the tests. But the important fact to note is that as we increase the blocksize to 84, we see an incline upwards in all the graphs, which means that the performance increases when blocksize is increased from 32 to 84. From blocksize 84 to 184, the performance is fluctuating, but after blocksize 256, the performance degrades for all the graphs. Similar observations were seen when the test was run on increasing number of processors.

Note that when tests were spawned on the ARC cluster, the HPL ran on all the processor cores that the node had to offer. If a test was run on 8 nodes, it means that the HPL ran on

128 cores, since each node in the ARC cluster has 16 cores as shown in Table 2. This is because, in this configuration, by default, the OMP_NUM_THREADS value is set to the total number of cores on each node.

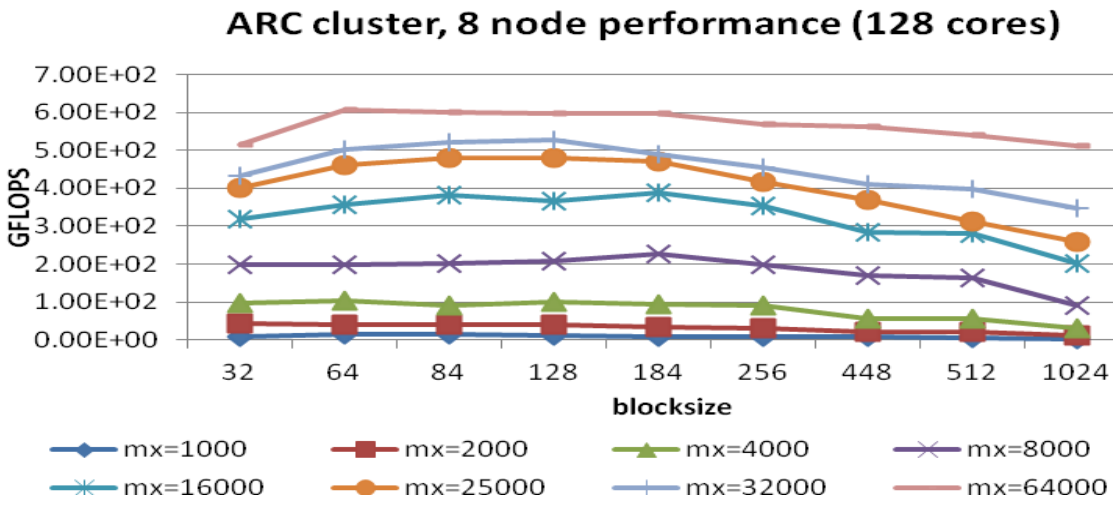

Figure 12: HPL Performance, 8 node (128 cores)

III. HPL with varying the process grid mappings (P and Q):

A different test with changing the grid mappings was done. This test could not be performed in the VCL configuration since, the VCL cluster was an experimental, statically configured cluster of four nodes. Since the requirement is that the product, P*Q should be less than or equal to the total number of MPI processes spawned, this test was not done on the VCL-GP cluster we used. This test gives a different context to understanding of the behavior of the ARC cluster. The results for 1,000, 32,000, 64,000 order matrices are shown in Figure 13, 14 and 15 (horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS) respectively. As we see, with matrix size 1,000, there was not much of a difference in the graphs when P and Q were changed. Although the average sustained performance of the combination P=2, Q=3 was better. It has been established and discussed on the HPL page [10], that P should be slightly less than Q. We have taken the argument to be true and did not experiment against the suggestion with P greater than Q.

As we increase the matrix size to 32,000 or 64,000, we see a difference. For order 64,000 matrix, the performance for P=2 and Q=4 was observed to be the best. It is interesting to note that both P and Q were powers of two. For larger matrices, the grid mappings which are a power of two perform better. Similar results were observed for 32,000 ordered matrix

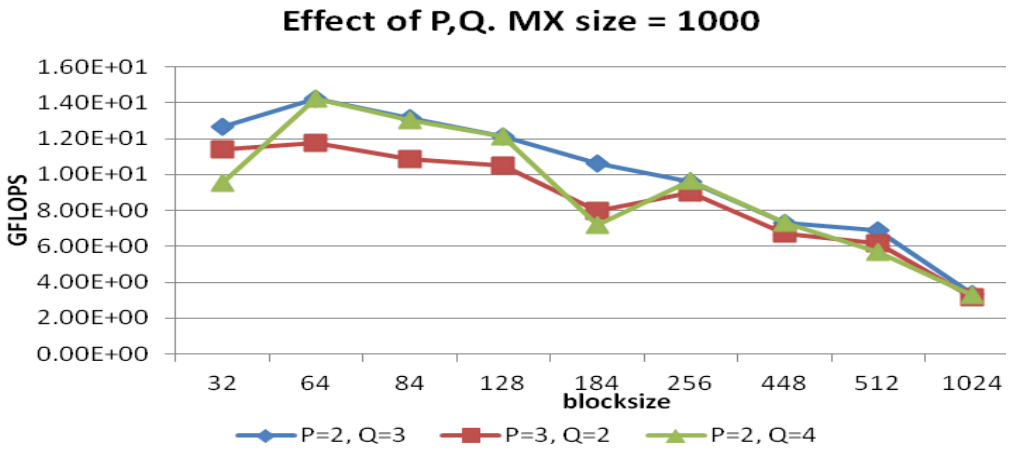

Figure 13: Effect of grid mappings, 8 node (128 cores), Matrix size=1000

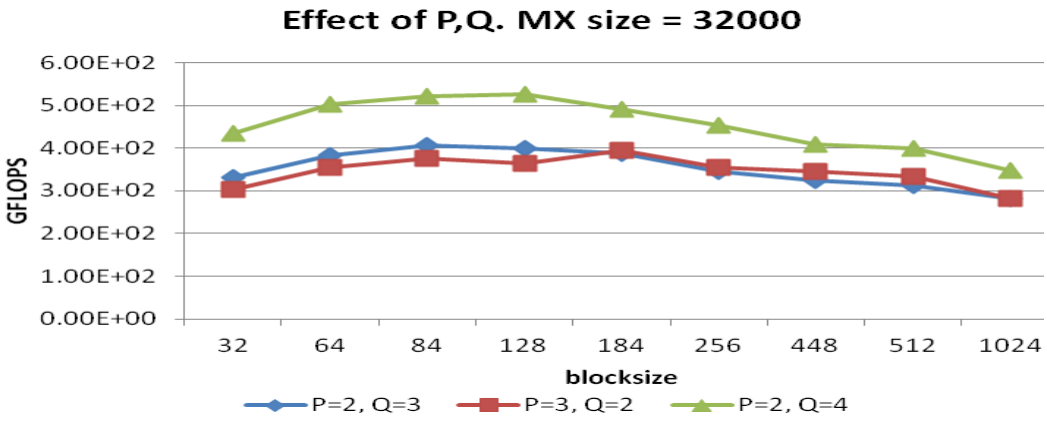

Figure 14: Effect of grid mappings, 8 node (128 cores), Matrix size=32000

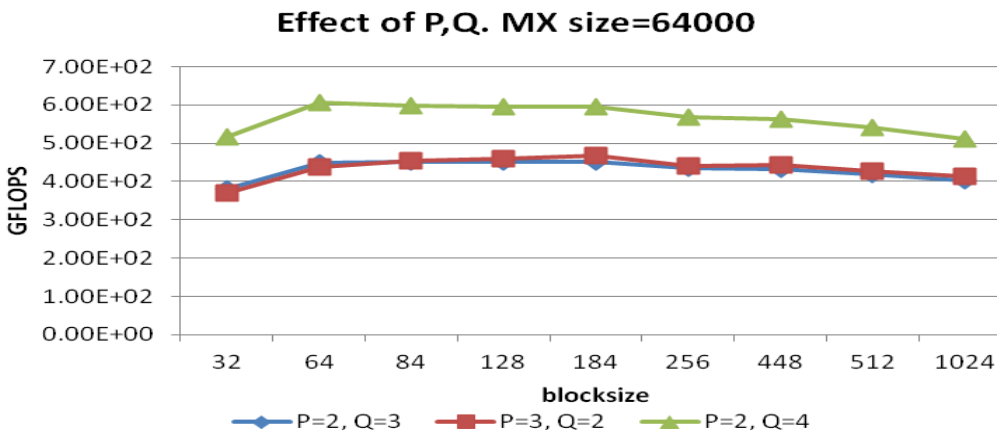

Figure 15: Effect of grid mappings, 8 node (128 cores), Matrix size=64000

IV. A simple analysis of the Ping-Pong benchmark results:

The latency-bandwidth test was done to measure the difference in latency or bandwidth when we increase the number of MPI processes. As shown in the Figure 16 (horizontal axis represent different configuration of processes and the left vertical axis represent the latency for 8 Byte messages in milliseconds and the right vertical axis represent the latency for 2 MB messages in milliseconds), the latency increases with increase in the total number of processes with the exception that the maximum Ping Pong latency for an 2 MB message decreases with increase in number of processes. Although there is slight increase or decrease in latency, the difference is not large enough to affect the performance. In fact, increasing the total number of processes increases the total performance on the ARC cluster. As seen in the bandwidth test in Figure 17 (horizontal axis represent different configuration of processes and the vertical axis represent the network bandwidth in MB/s), the bandwidth distribution is almost in harmony with the latency test. The minimum Ping Pong bandwidth increases slightly but overall the minimum bandwidth achieved is within the range 1940 to 2000 MB/s.

What this means is that when we increase the total number of processes, the minimum bandwidth achieved during large message passing between two nodes or in a naturally ordered ring of nodes or in a random ordered ring of nodes does not vary much. This means that as we scale to highly distribution applications, we will see a constant performance when

two nodes in a distributed system talk to each other directly or when ordered in a ring structure. The minimum, average and maximum bandwidth and latency reported by Ping Pong is shown in Table 3.

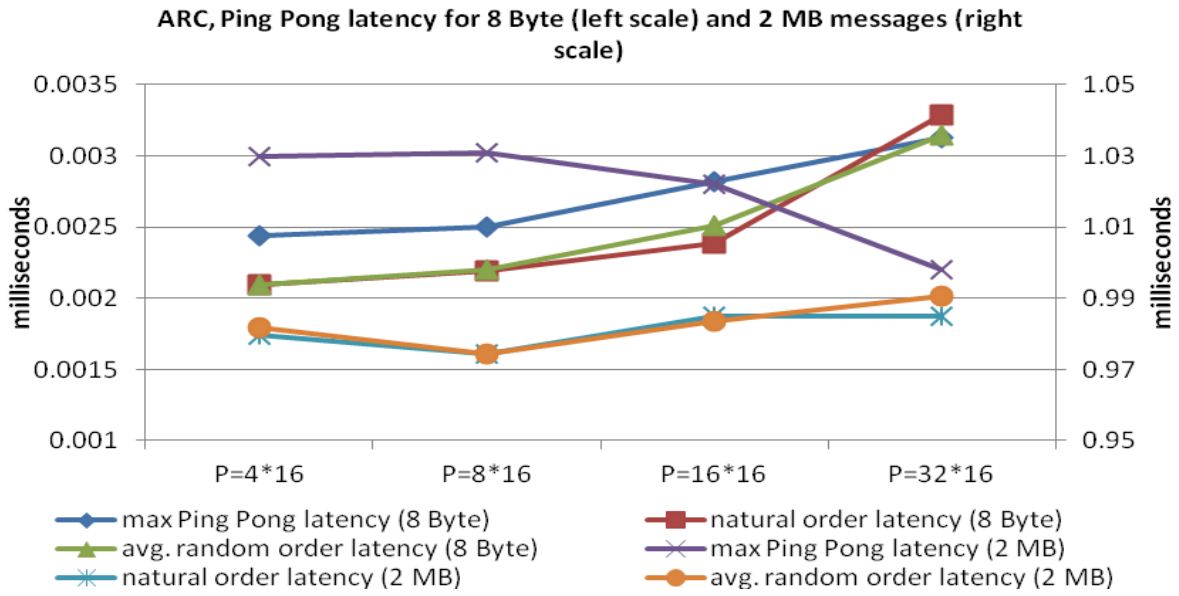

Figure 16: Ping pong latency, 8 node (128 cores, 8 Byte message and 2 MB message)

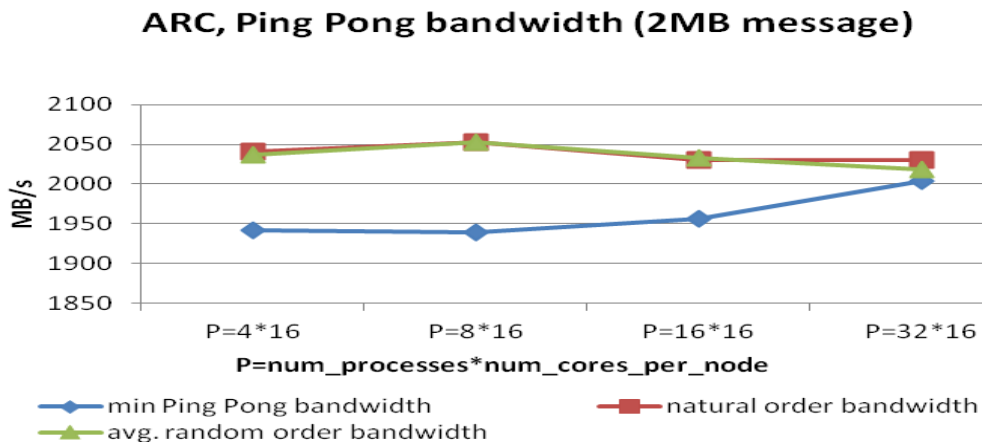

Figure 17: Bandwidth Test, 8 node (128 cores, 2MB message)

Table 3: Ping Pong Min/Avg/Max bandwidth and latency (ARC cluster)

Process configuration	2 MB message bandwidth (MB/s)			8 Byte Latency (milliseconds)			2 MB latency (milliseconds)		
(P=nodes*pr ocsper node)	min	avg	max	min	avg	max	min	Avg	max
P=4*16	1941.81	1990.99	2038.79	0.00218	0.0024	0.0024	0.98097	1.0048	1.03
P=8*16	1939.79	2043.05	2362.66	0.00212	0.0024	0.0025	0.84651	0.9819	1.031
P=16*16	1956.98	2084.58	2443.52	0.00225	0.0025	0.0028	0.81849	0.9623	1.022
P=32*16	2003.97	2318.17	2717.4	0.00231	0.0027	0.0031	0.736	0.8664	0.998

V. Interesting results and observations:

We also executed a test to explore the peak performance of the ARC cluster. Unfortunately we could only reserve a maximum of 32 nodes. As discussed previously, the peak performance was observed when the block size is in the range of 64 to 128. The interesting fact to note is that the peak performance achieved was 2.23 TFLOPS with 512 cores as shown in Figure 18 (the horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS). Assuming a balanced distribution of workload by HPL, this indicates that the performance of around 4.5 GFLOPS was achieved per core. With MPI traffic on the InfiniBand interconnects, we could achieve 4.5 GFLOPS per core. An interesting topic for further research would be to examine this with a larger number of processes. These tests would reveal whether the CPU utilization and the performance continue to increase or not.

Another interesting graph is shown in Figure 19 (the horizontal axis represent varying number of processor cores and vertical axis represents performance in GFLOPS). The graph shows peak performance achieved by HPL against the number of cores used. As we increase the processors in steps of power of two, we see a proportional increase in the HPL performance. If we extrapolate these results to the maximum number of nodes in the ARC cluster, we can reach to around 8 TFLOPS of peak performance on the ARC cluster with 1728 cores in total (108 nodes are taken into consideration). The 500th entry in the Top500 [44] list published in November 2011 is a HP ProLiant SL390s G7 Xeon 6C X5680 3.3 GHz,

GigE with 7236 cores which gives 50.9 TFLOPS. If the ARC cluster is expanded to around 800 nodes, we might be able to get to the Top500 list (this may or may not be true since the relative utilization drops as we go on increasing the number of processes).

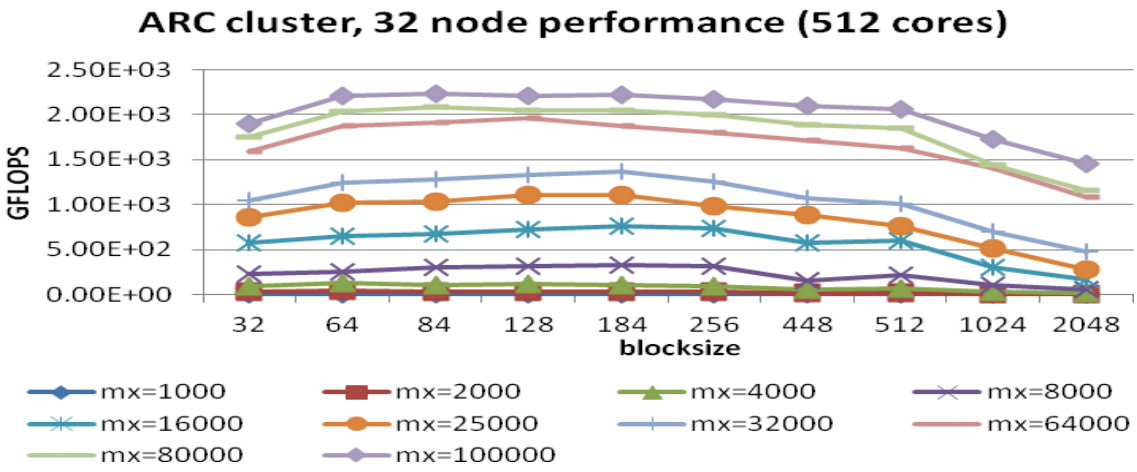

Figure 18: ARC cluster Peak Performance, 32 nodes (512 cores)

But this is not possible for an in-house cluster. If somehow we could attach 800 more nodes to the ARC cluster on the fly, requesting these resources from a cloud subsystem, we will have a supercomputer at NC State. Thus, more reasons to explore HPC in cloud.

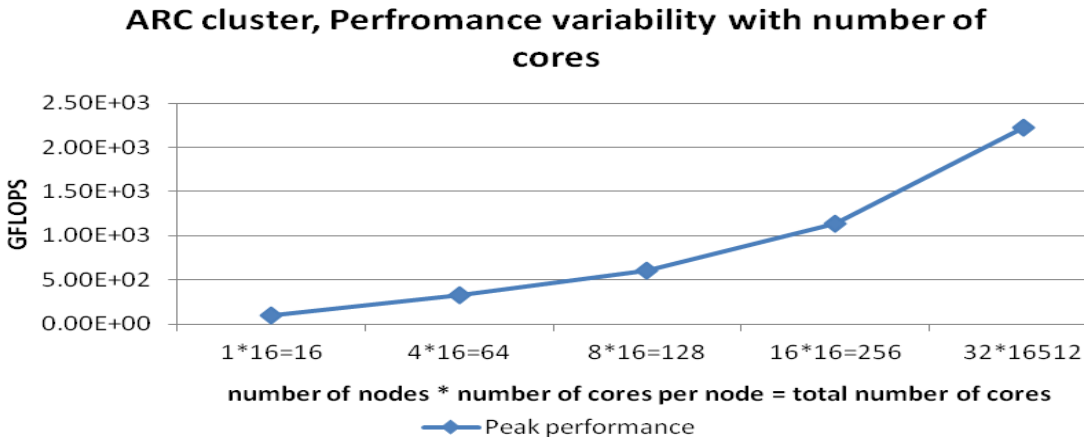

Figure 19: ARC cluster Peak Performance change with increase in number of cores

3.3.2 CPU and GPGPU cluster performance characterization (ARC, CPU+GPU)

LINPACK was initially developed in Fortran but later as LINPACK became widely used, a C version was developed. To test the performance of NVVIDIA GPGPU's, NVIDIA has developed a new CUDA based version of HPL. It is available from [21]. For the experiment we use the CUDA compatible HPL (version 1.3), cuBLAS library (version 4.1.21) and OpenMPI (version 1.5.4) libraries (more details are given at the beginning of section 3.3).

Even though ARC is a specialized cluster used for GPGPU workloads, the CPU characterization in the earlier section raises an important issue, is GPGPU a necessity? In the following section we discuss some of the performance numbers when HPL was run on CPU versus when HPL was run on CPU+GPU. The CUDA based HPL is tested on the ARC cluster. Configuration of the GPU card on a node in the ARC cluster is explained in Table 4 and 5.

Table 4: Configuration of a GPGPU card in the ARC cluster [3]

Cluster	ARC cluster
Queue	c2050
GPGPU Model	NVIDIA C2050 Tesla card
Microprocessor Architecture	Fermi
Clock Frequency (Core/Shader/Memory)	575/1150/3000 MHz
Number of GPU cards	1
Number of Cores/Stream processors	448
Hyper threading	NO
Theoretical Peak Performance	515 Gflops
L1 cache (per SM)	Configurable 16 KB or 48 KB
L2 cache	768KB
L3 cache	No
Memory	32 GB
Network	40G Infiniband/ PCI-E for CPU to GPU interconnect
Operating System	CentOS release 5.7 (Final)
OS architecture	X86_64

Table 5: Configuration of Tesla C2050 GPU card (used in the experiments) [91]

GPU	Fermi (c2050)
Transistors	3.0 billion
CUDA Cores	512
Double Precision Floating Point Capability	256 FMA ops /clock
Single Precision Floating Point Capability	512 FMA ops /clock
Special Function Units (SFUs) / SM	4
Warp schedulers (per SM)	2
Shared Memory (per SM)	Configurable 48 KB or 16 KB
L1 Cache (per SM)	Configurable 16 KB or 48 KB
L2 Cache	768 KB
ECC Memory Support	Yes
Concurrent Kernels	Up to 16
Load/Store Address Width	64-bit

I. HPL performance on a single node with CPU+GPU:

As seen in the graph in Figure 20 (the horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS), a single node CPU+GPU could deliver around 268 GFLOPS. From the results in Figure 11 and 20, we observe that the performance achieved by a single core CPU plus a 448 core GPU together, is more than twice when compared to a standalone CPU system with 16 cores. Thus a GPGPU is important when compared to processing power contained in a single node.

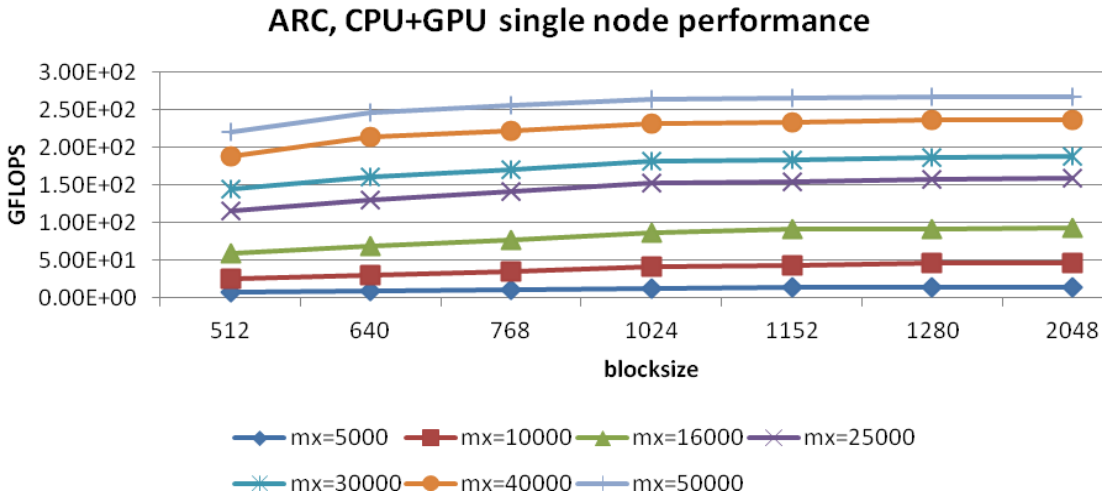

Figure 20: HPL performance on ARC cluster single node CPU+GPU (1 node, 1 core, 1GPU-448 cores)

II. HPL with varying Blocksize and Matrix size (NB, N):

Again in this case, as shown in Figure 21 (the horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS), as we go on increasing the matrix size the performance increases. This is because of the linear scaling of the HPL algorithm. On a CPU only configuration, in the previous section, as the block size was increased, the performance increased until a point of maximum performance is reached, and then the performance drops. But as we observe in the case of CPU+GPU, increasing the blocksize increases the performance. An important observation done here is that the performance increases until blocksize 1152 and drops after blocksize 1280 for almost all matrix sizes. This is greater than the drop point for CPU only configuration. In depth, understanding of the GPGPU cache architecture and lots of memory profiling needs to be done to explain this behavior. This is beyond the scope of the report and is left for further research.

Extreme tests were conducted to check whether blocksize affects the performance in terms of a CPU+GPU configuration. The maximum blocksize allowed for HPL was 2048 and hence the test could not go beyond the results in Figure 21 and 22. Figure 22 (the horizontal axis represent varying matrix size and vertical axis represents performance in GFLOPS) shows that the graph for blocksize 1280 illustrates maximum performance. The drop in performance was seen for blocksize 2048. The drop was significantly seen in larger matrices. For example, in this scenario for the 80000 order matrix, the performance of HPL significantly drops beyond blocksize 1280 (Figure 21). In case of the 40000 order matrix, the performance takes a hit after blocksize 1152. Although the tests were successful it is worth noting that these tests show inconsistent behavior with 8 or more processes on the ARC cluster. Although it was observed that when we reserved 16 nodes, we could run the 8 node test (Nodes were requested using the command “qsub -I -q c2050 -l nodes=16:ppn=1”). Similarly for a 16 node test, we had to reserve more than 16 nodes (we requested 32 nodes once and 24 nodes the second time for the jobs to execute). Reasons are not clear and this needs more investigation. It could be a configuration issue in the ARC GPU queue or a bug in the HPL-CUDA code.

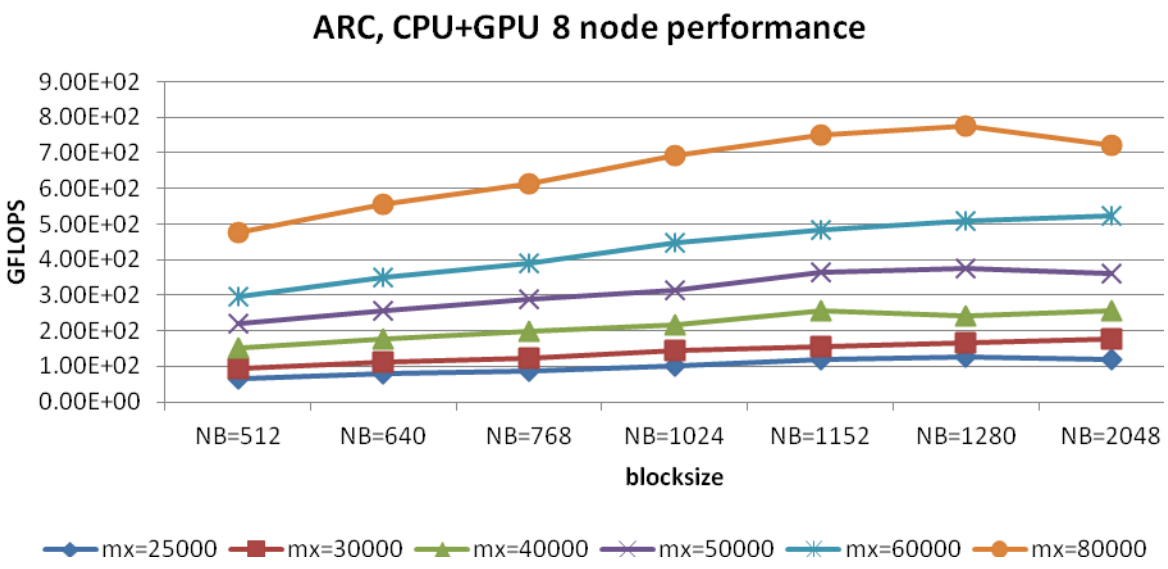

Figure 21: CPU+GPU performance, 8 node (8 cores + 8 GPU)

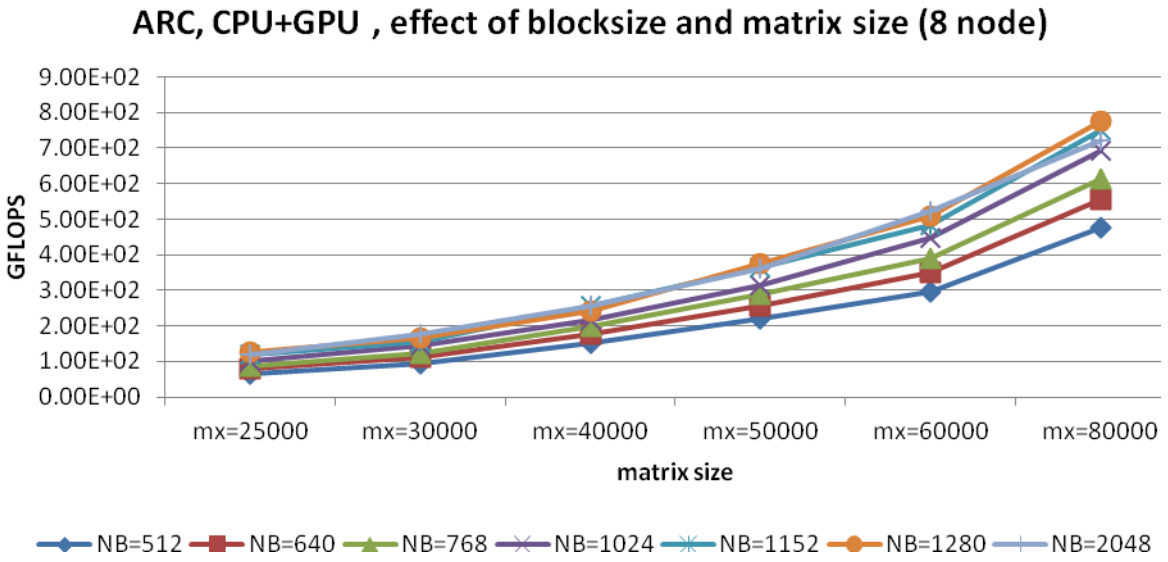

Figure 22: Effect of blocksize and matrix size, 8 node (8 cores + 8 GPU)

III. Interesting Observations:

The maximum peak performance was measured with increasing the total number of MPI processes. The result observed was similar to the case of "CPU only" configuration, except we see a drop in performance. We have speculated that the drop in performance is because

we used 100000 order matrix for 8 nodes as well as 16 nodes. We expect 16 nodes to perform better with larger matrices (Tests for larger matrices could not be completed due to memory limitations. Errors due to memory allocation were seen. A 16 node test for 120000 order matrix did not complete before the allotted wall time for the job).

When we compare the results form Figure 23 (the horizontal axis represent varying number of nodes and vertical axis represents performance in GFLOPS) and Figure 19, it was observed that we could reach 2.23 TFLOPS with 16 nodes (512 CPU cores) but in case of CPU+GPU we could not reach the 1 TFLOPS boundary even with 16 nodes. We repeated this test to make sure that the analogy matches every other time. But it was very difficult to repeat this test since the 8 node and 16 node tests were running into timeout, hang and memory issues. These tests were not consistent. The results form a second test at time T2 (a week after T1) is shown in Figure 24 (the horizontal axis represent varying number of nodes and vertical axis represents performance in GFLOPS). As seen here, the performance increases with increase in the number of nodes. The maximum performance observed was 1.34 TFLOPS and is much better than 784 GFLOPS as observed at time T1. The behavior is not consistent since the tests failed several times as explained and multiple attempts were made to get two different set of results. Note that T1 and T2 were a week apart. We assume that the instability is because of the optimal split calculated by the algorithm [22]. If the split is suboptimal at different times, the performance can degrade. This might be one of the reasons for the variability but we do not have a deep and thorough explanation for the instability and is an open question left for future work.

Earlier, we made an argument that a single node GPU+CPU configuration outperforms a “CPU only” configuration. The observations here raise a new question, “is GPGPU a good option in a clustered environment?” In chapter 5 of this report, we have discussed options for using multi-GPU in a clustered environment which could possibly prove to be more stable, more consistent and use only GPU for computations and leave the CPU to take the role of a communicator.

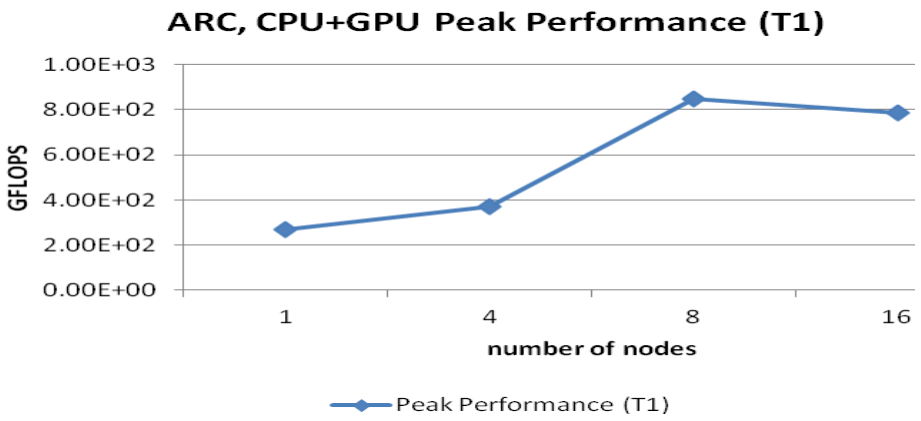

Figure 23: Effect of number of processes on HPL performance at time T1

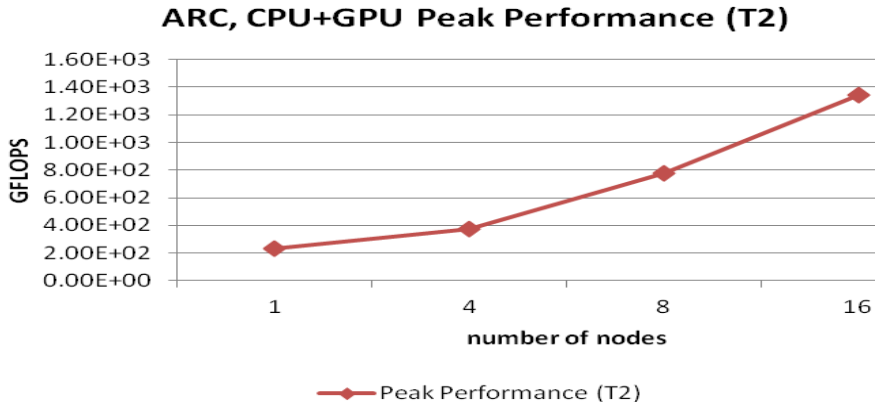

Figure 24: Effect of number of processes on HPL performance at time T2

3.3.3 Performance of a compute intensive application, Lattice QCD

"Lattice QCD [58, 59] (quantum chromodynamics) is the lattice discretized theory of the strong force, that which binds together quarks in the nucleon. In lattice QCD, the propagation of quarks is given by the inverse of the Dirac operator, which is a large sparse matrix [58, 59]". Hence, many systems of linear equations must be solved involving this matrix; it is this requirement that makes lattice QCD a grand challenge subject. "In such linear solvers, the application of the Dirac operator to a vector is the most compute intensive kernel. Quda is an initial exploration of how best to implement these solvers on a single GPU system" [58, 59]

“Quda is a library written for performing calculations in lattice QCD on graphics processing units (GPUs) using NVIDIA's "C for CUDA" API [15]. The current release includes optimized solvers for the following fermion actions:

- Wilson
- Clover-improved Wilson
- Twisted mass
- Improved staggered (asqtad or HISQ)
- Domain wall

Mixed-precision implementations of both CG and BiCGstab are provided, with support for double, single, and half (16-bit fixed-point) precision. The staggered implementation additionally includes support for *asqtad* link fattening, force terms for the *asqtad* fermion action and one-loop improved *Symanzik* gauge action, and a multi-shift CG solver” [58, 59]. The basic function of the Quda library is to solve a system of equations. This is similar to a LINPACK except that this is very tightly coupled to the Wilson-Dirac equation and does not solve a general system of linear equations.

The Quda library is a self tunable library. The library needs to be tuned before it is used to solve the Wilson-Dirac equation. The library on first build is not properly tuned. Tuning the library is necessary to get maximum performance. The tuning takes place by issuing a “make tune” command. The tuning process involves executing a basic blas_test on the node on which the final tests are going to be executed. The blas_test will spawn threads with different grid configuration to find an optimum configuration. For compute intensive applications that use CUDA, a special purpose BLAS library called CUBLAS is used. The tuning process calls a certain CUBLAS API's to understand the system and provide proper optimizations. The updates are written to the “blas_param.h”. Essentially it understands the optimum configuration. Now we need to rebuild the library to reflect the optimum parameters.

We have performed the Quda experiments on the configurations described in Table 2, 4, and 5. The Quda library version used was QUDA v0.3.2 (18 January 2011) available from [58].

The compiler options were kept default in the Makefile. No changes were made to the make.inc except for the path to CUDA libraries. The compiler optimization used was –O3. Following are a series of graphs shown in Figure 25, 26 and 27 representing half-precision, single precision and double precision performance numbers. Note that these are shown as a 3D view of a graph with two units represented on the vertical axis. The vertical axis represents performance in GFLOPS as a blue colored graph and memory bandwidth in GiB as a brown colored graph. The horizontal axis represents different API calls called by the blas_test. It shows the tests or API calls that blas_test runs to find the optimum parameters for half precision, single precision and double precision BLAS operations. The three graphs represent the tuning performance for half, single and double precision respectively. As we can see from the graphs, that as the precision increases, the total performance of the computations drops. Each higher level of precision almost takes double the time or performs half as well as the previous less precision arithmetic. Although it is observed that the memory bandwidth (GiB) requirements for all three precision floating point operations remains the same. This is an indication that the performance will always be memory bandwidth limited

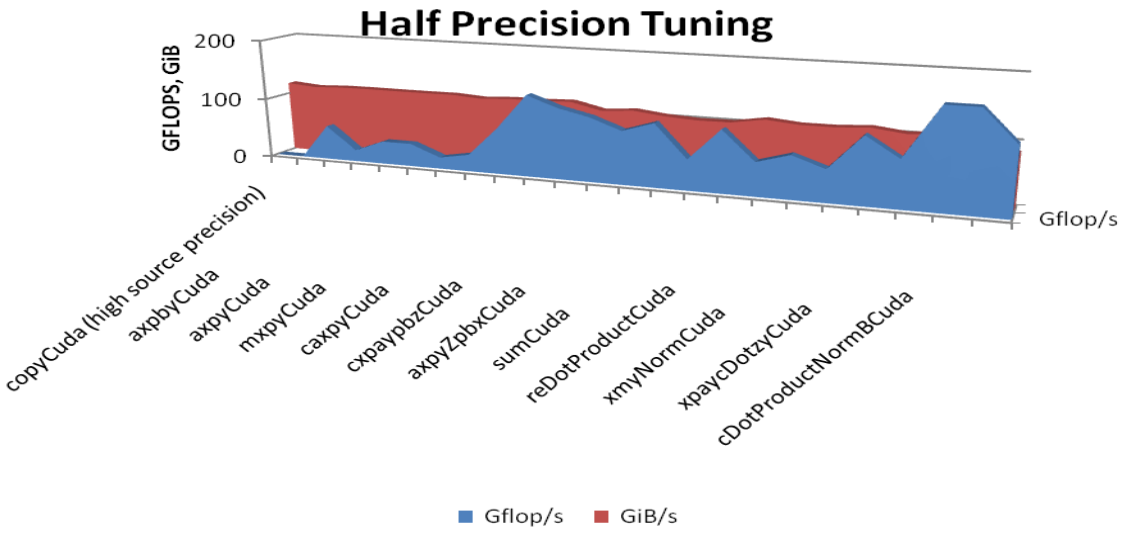

Figure 25: Tuning Half Precision floating-point operations

.

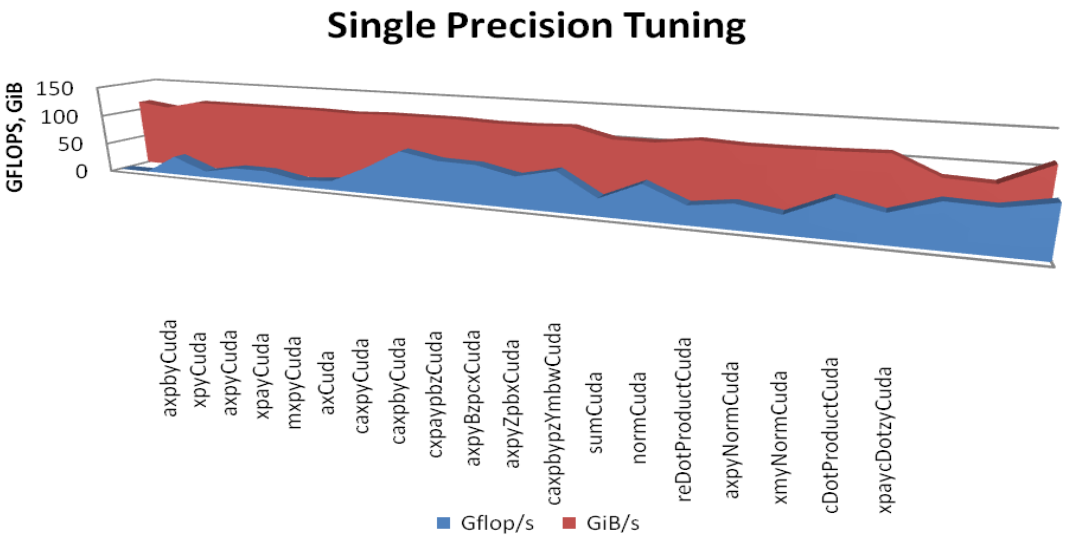

Figure 26: Tuning Single Precision floating-point operations

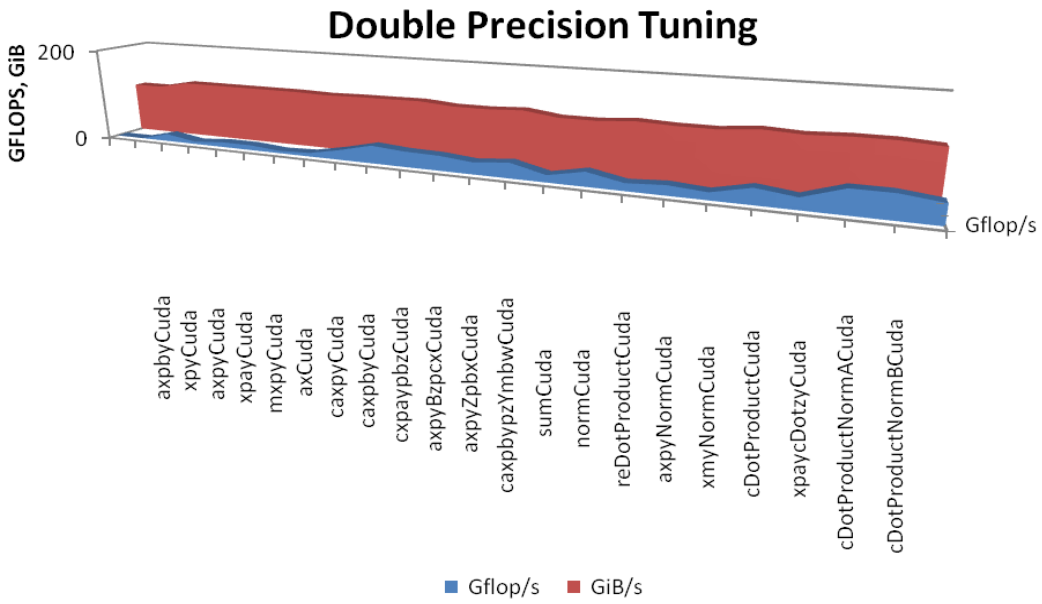

Figure 27: Tuning Double Precision floating-point operations

Performance results for CG and BiCGstab solvers (Wilson-Dirac Matrix):

From the results it was observed that the mixed precision operations to solve the Wilson-Dirac matrix equations work well. The maximum performance observed was 209 GFLOPS

on the wilson_invert_test, which uses the BiCGstab solvers. Figure 28 and 29 represent the Dslash and the Invert tests. It is again a 3D view of a graph with two units represented on the vertical axis. The vertical axis represents performance in GFLOPS as a blue colored graph and memory bandwidth in GiB as a brown colored graph. The horizontal axis represents the different Dslash/Invert tests.

Figure 30, shows a comparison of three different benchmarks Quda, High Performance LINPACK and 3D Stencil Codes [73] which were run on a single GPU node on the ARC cluster. It was observed that HPL achieved maximal performance compared to the other applications. But it is worth to note that HPL uses CPU+GPU to solve the set of dense linear equations Ax=B. Quda, which performs second best, was very near to the HPL performance. This explains that, taking advantage of the underlying architecture to its full potential gives better performance. The Quda tests indicate that using mixed precision arithmetic to solve a double precision computational problem by actually doing single precision operations achieves better performance than just single precision or double precision alone.

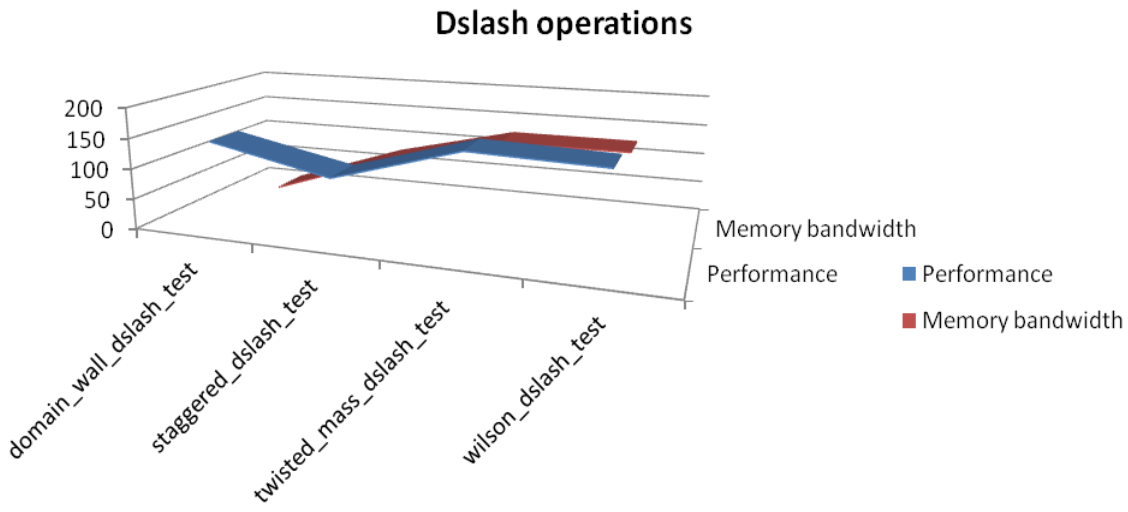

Figure 28: Performance of Dslash operations

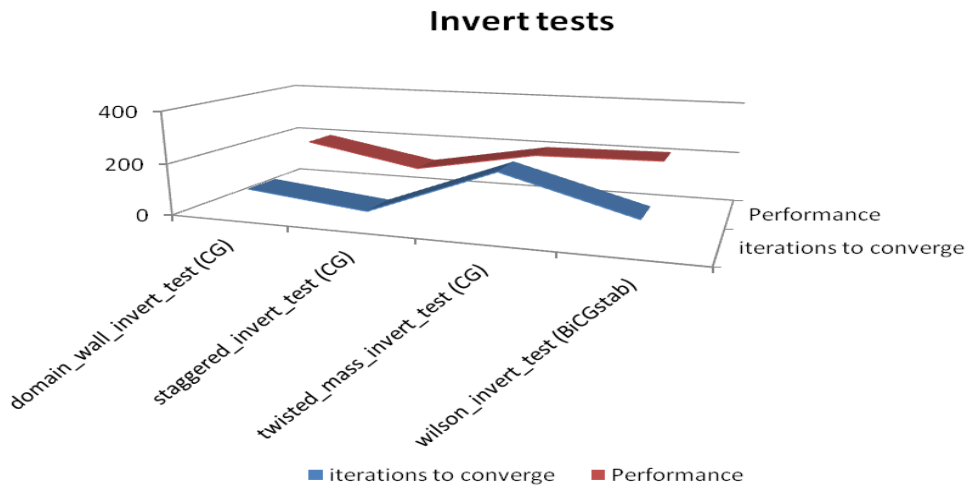

Figure 29: Performance of Invert operations

The reason is quite well understood in the Fermi architecture [14]. Fermi has 32 cores in a single streaming processor and can achieve 32 single precision floating point operations per cycle. On the other hand, a single stream processor can provide for just 16 double precision floating point operations per clock cycle. Because of this limitation, the performance of pure double precision arithmetic is degraded or statistically saying, it can be considered half of the single precision performance. In practical scenarios where the underlying ISA (Instruction Set Architecture) does not support vector operations (for eg: SSE instructions), the double precision operations perform less efficiently than half of the single precision performance. The Quda library takes advantage of both these precision operations to achieve maximum performance. "Quda library introduces a new method for using mixed precision in the context of Krylov solvers, repurposing the reliable updates scheme [59]". The results confirm the above statement and shows that the Quda library can solve the system of equations related to Lattice QCD efficiently. We run the High performance LINPACK using double precision arithmetic (8 byte words) since that is the standard followed by the HPL and top500 [57] community; the single precision numbers for HPL were not explored and hence are not shown on the graph. The 3D stencil codes performance vary according to the GPU card used. On Geforce GTX 480 the results were 203 GFLOPS single precision and 86 GFLOPS double precision but on the Tesla C2050 cards the results were 157 GFLOPS single precision and 97 GFLOPS double precision [73].

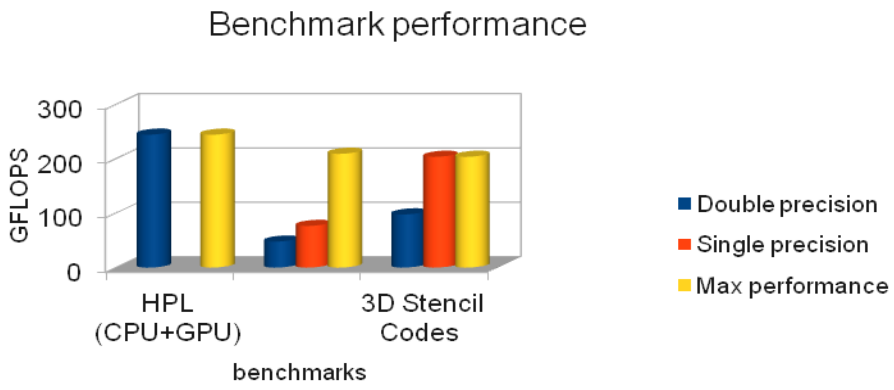

Figure 30: Performance comparison of benchmarks on single GPU

3.4 Performance evaluation of compute jobs on VCL-HPC (Henry2) cluster nodes

The Henry2 cluster is the hardest to evaluate. This is because it has a combination of old servers as well as new servers. As hardware changes, so does the limits on performance. In addition, the Henry2 cluster has the following combination of compilers and Math Libraries. It has Intel compilers (icc, icpc, ifort) which can work with Intel MKL (Math Kernel Library) libraries [97] as well as the GotoBLAS2 [90] math library. GNU (gcc) compilers are installed and can work with Intel MKL as well as GotoBLAS2 math libraries. Furthermore, there are different compiler environments for different interconnects. InfiniBand has different environment ("source /home/gwhowell/scripts/mvapich-intel/int101_mvapich.csh will update the environment to call the icc compiler compatible with the machines that are connected to the InfiniBand network" [2]). A list of compilers and Math libraries used for the experiment are noted in Table 6 below.

Table 6: Compilers and Math Libraries on Henry2

Compiler	Math Library	Interconnect
Intel (icc (ICC) 10.1)	Intel MKL (9.1.023)	1Gb/s Ethernet
Intel (icc (ICC) 10.1)	GotoBLAS2 (1.13)	1Gb/s Ethernet
GNU (gcc (GCC) 4.1.2)	Intel MKL (9.1.023)	1Gb/s Ethernet
GNU (gcc (GCC) 4.1.2)	GotoBLAS2 (1.13)	1Gb/s Ethernet
Intel (icc (ICC) 10.1)	Intel MKL (9.1.023)	InfiniBand (40Gb/s)
Intel (icc (ICC) 10.1)	GotoBLAS2 (1.13) (not tested)	InfiniBand (40Gb/s)
GNU (gcc (GCC) 4.1.2)	Intel MKL (9.1.023) (not tested)	InfiniBand (40Gb/s)
GNU (gcc (GCC) 4.1.2)	GotoBLAS2 (1.13) (not tested)	InfiniBand (40Gb/s)

The following gives a good idea about the different compilers and the method to use them.
“GNU compilers:
The gnu compilers are available in the default path and are invoked with the cc and f77 commands for the C/C++ and Fortran77 compilers respectively. gfortran invokes a gnu fortran90 compiler. Open source codes typically are rather easy to compile with gnu compilers [2].

To use the Intel or PGI compilers it is necessary to properly configure some environment variables and paths. The following description gives a brief idea of the environment. As a convenience an alias - add - has been created for tcsh users to set up the environment for various software packages [2].

Intel Compilers:
Intel compilers produce efficient executables for Intel blades and allow linking to the intel MKL math libraries. Linking object files to produce an executable sometimes requires shuffling the order in which the object files are listed, or listing object files multiple times. 64 bit Intel compiled executables for AMD blades encounter run time errors [2].

On 64 bit login nodes, run the commands *add intel_mpich2_hydra-101* to access or link the mpich2 libraries. To access or link to the mpich1 libraries (on 32 bit or 64 bit) the command *add_intel* works. The above commands set the environment for Intel 10.1 compilers [2]. To use older compiler versions refer to [2] for more information.

PGI Compilers:
Portland group compilers are "user friendly" and produce efficient executables. There is run-time check for a valid license, so PGI compiled codes are valid only on machines with access to PGI licenses. Some useful 64 bit numeric libraries for pgi compilers can be found in /usr/local/apps/acml/acml4.3.0 [2].

Similar to the Intel compilers, the *add pgi* command will set the necessary environment variables for 32 bit or 64 bit login nodes. For 64 bit login nodes, the command *add pgi64_hydra* enables use of mpich2 hydra, which avoids some run time error messages (net_send ..) associated with the mpich1 libraries invoked by *add pgi* command. The above commands set the environment for PGI 10.5 compilers [2]."

As mentioned above, it is worth to mention that the Intel compilers were very particular of the order in which the libraries were included. For e.g.:
The correct order in Makefile is:
"LAdir = /usr/local/intel/mkl91023/lib/em64t
LAinc =
LAlib = $(LAdir)/libmkl_lapack.a $(LAdir)/libmkl_em64t.a $(LAdir)/libguide.a $(LAdir)/libfftw2xc_intel.a $(LAdir)/libmkl_solver.a"
Any other order in "LAlib" did not work and gave undefined reference errors during compilation.

Performance characterization of Henry2 cluster (CPU performance):
To provide a performance characterization of the Henry2 cluster, the LINPACK benchmark was run with different configurations as noted in the tables below. The configurations used for different tests were different. This is because of difference in the compiler and network interconnects. We have used the queue "single_chassis", which tries to keep all assigned cores within a single IBM blade center chassis (also the default queue), for all of the tests except whenever InfiniBand is required. For InfiniBand configurations we have used the queue "standard_ib".

The following tables (7 to 12) provide the configuration details. For each graph or figure in this section we have mentioned the configuration used. Since the user does not have a lot of control over the configurations, the jobs are executed on the configurations as decided by the LSF scheduler. Although the following tests were executed on the configurations mentioned, it is not necessary for subsequent similar tests to get the same resources. The multi-node tests

illustrated below run on configurations that are similar but this is not promised by the LSF scheduler. For each test below, a typical configuration as provided by LSF is mentioned. We have not requested any special configurations. This illustrates that the user needs to be aware of the intricacies of the pre-defined cluster to get better performance.

Table 7: CPU Configuration 3: A node in Henry2 cluster

Cluster	Henry2
Queue	single_chassis
Processor Model	Intel Xeon L5335
Microprocessor Architecture	core
Clock Frequency	2 GHz
Number of Processors	2
Number of Cores	8
Hyper threading	NO
L1 cache	32k-Instr, 32K-Data per core
L2 cache	256K-Unified per core
L3 cache	8M-Unified per processor
Memory	24 GB
Network	1 Gb/s Ethernet
Operating System	CentOS release 5.4 (Final)
OS architecture	X86_64

Table 8: CPU Configuration 4: A node in Henry2 cluster

Cluster	Henry2
Queue	single_chassis
Processor Model	Intel Xeon E5540
Microprocessor Architecture	Nehalem-EP
Clock Frequency	2.53 GHz
Number of Processors	2
Number of Cores	8
Hyper threading	NO
L1 cache	32k-Instr, 32K-Data per core
L2 cache	256K-Unified per core
L3 cache	8M-Unified per processor
Memory	24 GB
Network	1 Gb/s Ethernet
Operating System	CentOS release 5.4 (Final)
OS architecture	X86_64

Table 9: CPU Configuration 5: A node in Henry2 cluster

Cluster	Henry2
Queue	standard_ib
Processor Model	Intel Xeon E5649
Microprocessor Architecture	Westmere-EP
Clock Frequency	2.53 GHz
Number of Processors	2
Number of Cores	12
Hyper threading	NO
L1 cache	32k-Instr, 32K-Data per core
L2 cache	256K-Unified per core
L3 cache	12M-Unified per processor
Memory	48 GB
Network	40Gb/s InfiniBand
Operating System	CentOS release 5.4 (Final)
OS architecture	X86_64

Table 10: CPU Configuration 6: A node in Henry2 cluster

Cluster	Henry2
Queue	single_chassis
Processor Model	Intel Xeon E5520
Microprocessor Architecture	Nehalem-EP
Clock Frequency	2.27 GHz
Number of Processors	2
Number of Cores	8
Hyper threading	NO
L1 cache	32k-Instr, 32K-Data per core
L2 cache	256K-Unified per core
L3 cache	8M-Unified per processor
Memory	24 GB
Network	1 Gb/s Ethernet
Operating System	CentOS release 5.4 (Final)
OS architecture	X86_64

Table 11: CPU Configuration 7: A node in Henry2 cluster

Cluster	Henry2
Queue	single_chassis
Processor Model	Intel Xeon X5650
Microprocessor Architecture	Westmere-EP
Clock Frequency	2.67 GHz
Number of Processors	2
Number of Cores	12
Hyper threading	NO
L1 cache	32k-Instr, 32K-Data per core
L2 cache	256K-Unified per core
L3 cache	12M-Unified per processor
Memory	36 GB
Network	1 Gb/s Ethernet
Operating System	CentOS release 5.4 (Final)
OS architecture	X86_64

Table 12: CPU Configuration 8: A node in Henry2 cluster

Cluster	Henry2
Queue	single_chassis
Processor Model	Intel Xeon E5504
Microprocessor Architecture	Nehalem-EP
Clock Frequency	2 GHz
Number of Processors	2
Number of Cores	8
Hyper threading	NO
L1 cache	32k-Instr, 32K-Data per core
L2 cache	256K-Unified per core
L3 cache	4M-Unified per processor
Memory	16 GB
Network	1 Gb/s Ethernet
Operating System	CentOS release 5.4 (Final)
OS architecture	X86_64

Following are some more configuration details used during the tests performed on the Henry2 cluster.

LSF (Load Share Facility) resource scheduler version: 6.1 (Henry2 cluster)

HPL code available from [105]

GotoBlas2 math library: Optimized GotoBLAS2 libraries version 1.13 available from [90]

GNU, CC FLAGS: -lm -fomit-frame-pointer -O3 -funroll-loops -Wall

Intel, CC FLAGS: -fomit-frame-pointer -O3 -funroll-loops -Wall

I. HPL performance on a single node in Henry2 cluster:

In this scenario we performed single node tests on the CPU configuration 4 and configuration 5. As is the case in all the above scenarios, a single node with 8 cores gave a max performance of around 51.5 GFLOPS on configuration 4. In Figures 31 to 37 we follow the following graph configuration. The horizontal axis represent varying blocksize and vertical axis represents performance of HPL in each of the configurations indicated in GFLOPS.

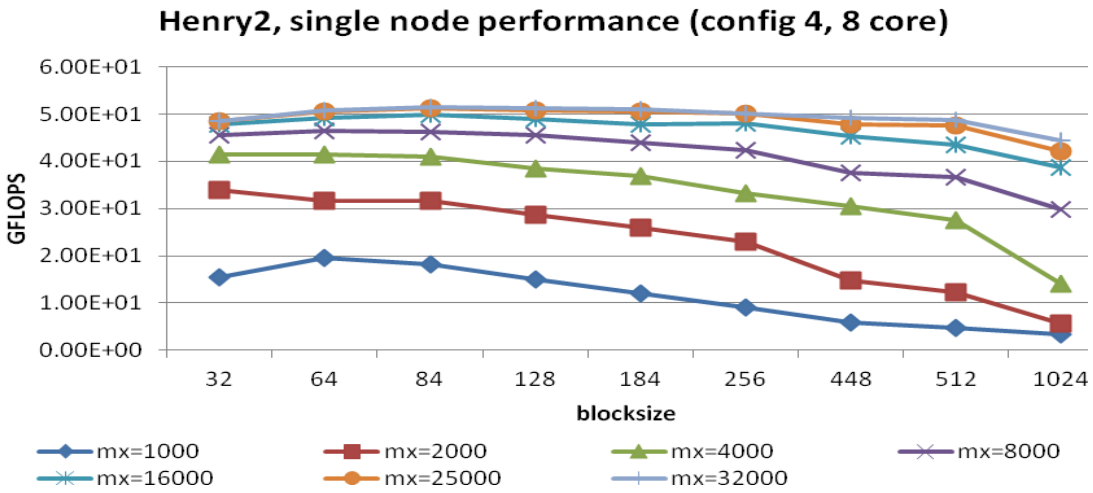

Figure 31: HPL performance on single node, (1 node, 8 cores, CPU configuration 4)

On configuration 5, we got a max performance of 52.6 GFLOPS. A slight increase in performance compared to configuration 4 is seen. Even though configuration 5 has 12 cores, we ran the test on just 8 cores so that we could have similar results to compare with configuration 4. Note that in the case of nodes from Henry2 cluster, we have to spawn 8 MPI

processes to put all the cores in a node to work. The option of using a single process and increasing the value of OMP_NUM_THREADS can be tried but the results were diminishing in a few cases (e.g. Figure 37).

With further investigation, we found that the Intel MKL library that we were using was not performing well with multithreading. We do not have enough data to confirm this behavior. More specific tests need to be performed to confirm this behavior.

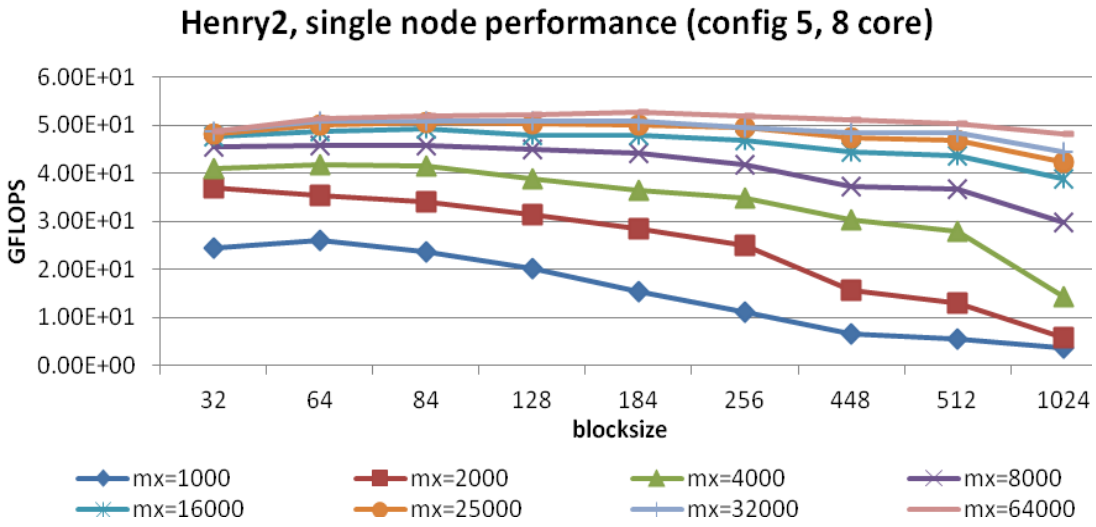

Figure 32: HPL performance on single node, (1 node, 8 cores, CPU configuration 5)

II. HPL with varying Matrix size and block size for various compilers (N, NB):

Figures 33 to 37 show the performance observed for different combination of compilers and the Math libraries. Note that we do not have results for all the combinations. This is because we did not request special configurations.

The results presented in this section are very specific to the compiler used and the math library used. For the Intel compiler and GotoBLAS2 math library in Figure 35, we observe that the performance increases as we go on increasing the matrix size as well as the blocksize. The maximum performance was observed for a 64,000 order matrix. The graph for 64,000 order matrix is incomplete since the test did not complete in time. The interesting fact

that is observed here is that although the peak performance achieved was comparable to other tests, the overall performance in this case is not good. The performance was suboptimal as shown in Figure 35. Performance was very less for the combination GNU compiler/GotoBLAS2 math library as illustrated in Figure 37. In Figures 33, 34 and 36, the performance observed was good and follows the practical limit, as explained in the section above, the practical limit for 8 cores is 51.5 GFLOPS for CPU configuration 4 and 52.6 GFLOPS for CPU configuration 5.

The results from all the above combinations show that, the GotoBLAS2 library did not work well on the Henry2 cluster. The same GotoBLAS2 library gave the performance of nearly 2.23 TFLOPS on the ARC cluster. The difference between the Henry2 cluster and the ARC cluster test was the use of different compilers (Intel/GNU) and MPI libraries (MPICH2) on henry2 cluster.

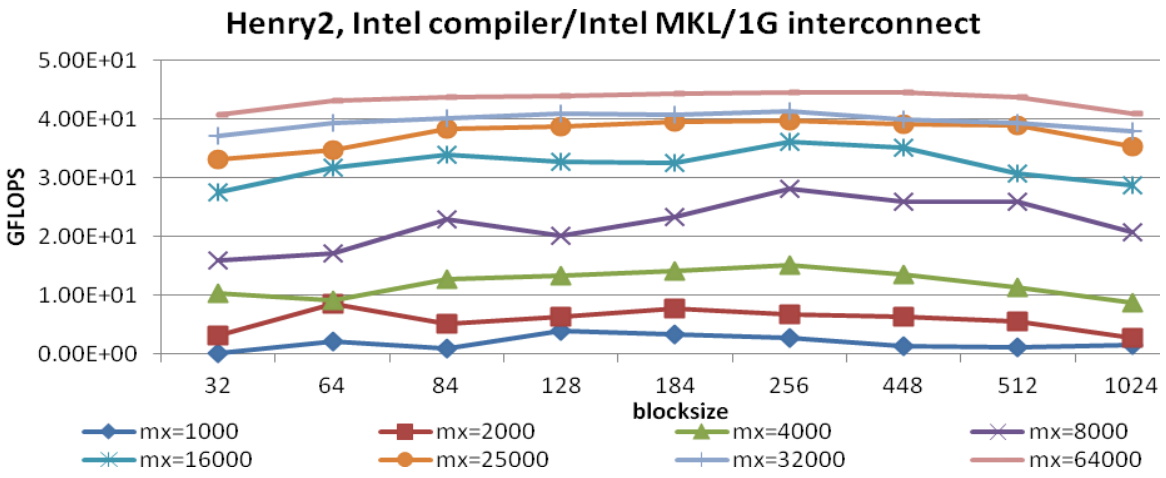

Figure 33: HPL performance characteristics Intel compiler/Intel MKL/1G interconnect (4 node, 8 cores, CPU configuration 6)

Form the discussion, it could be informally said that GotoBLAS2 does not work well on the Henry2 cluster because of either the compiler or the MPI libraries. The graph in Figure 38 clearly depicts the situation and explains why it is so hard to get performance form the Henry2 cluster. Note that some of the graphs lines in Figures 33 to 37 are not complete. This is because the tests were slow enough not to complete and were forced closed by the

scheduler. Most of these tests took long time to run. We do not have sustained results over time. The results illustrated here are the output of a job run with the typical configuration as mentioned. The graph in Figure 38 shows a comparison of the different compilers and math libraries.

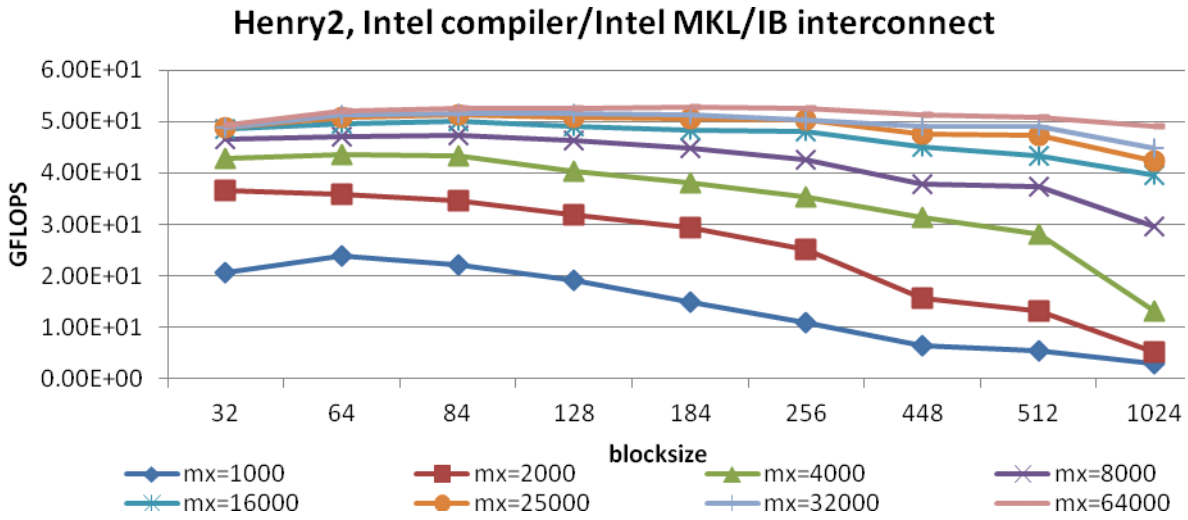

Figure 34: HPL performance characteristics Intel compiler/Intel MKL/IB interconnect (4 node, 8 cores, CPU configuration 5)

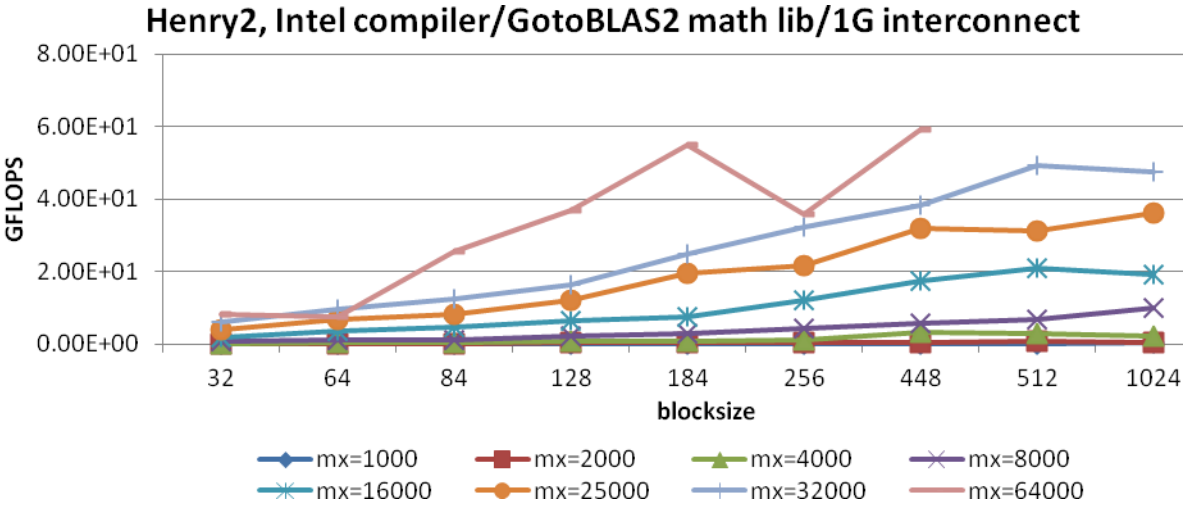

Figure 35: HPL performance characteristics Intel compiler/GotoBLAS2 math library/1G interconnect (4 node, 8 cores, CPU configuration 6, P=2. Q=3)

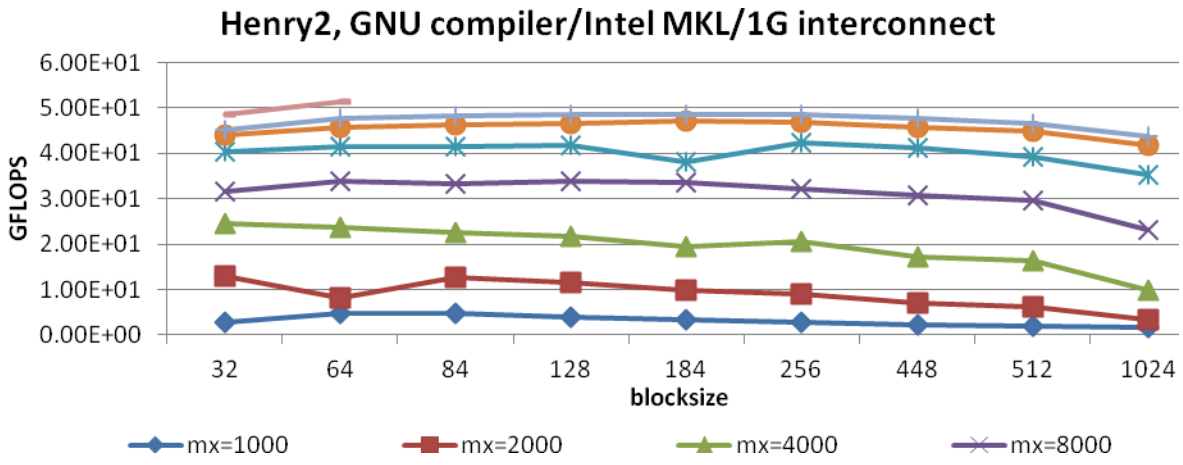

Figure 36: HPL performance characteristics GNU compiler/Intel MKL/1G interconnect (4 node, 8 cores, CPU configuration 7)

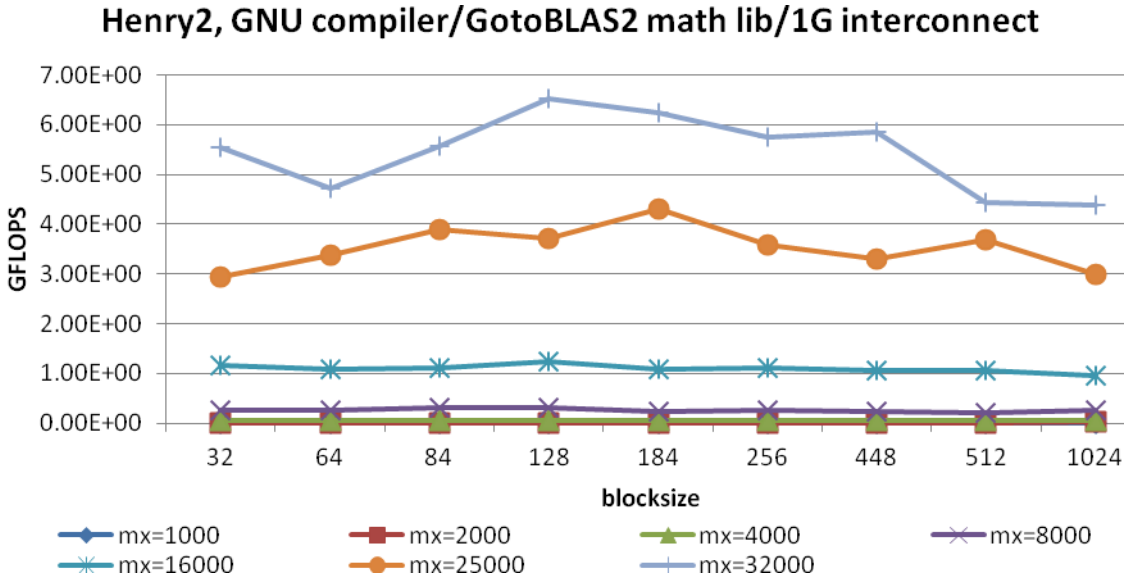

Figure 37: HPL performance characteristics GNU compiler/GotoBLAS2 math library/1G interconnect (4 node, 8*4 cores, P=2, Q=3, OMP_NUM_THREADS=4, CPU configuration 8)

The explanation of each configuration on the horizontal axis for Figure 38 is as follows.

intel_goto_1G → Intel compilers, GotoBLAS2 library, 1G interconnect

intel_intel_IB → Intel compiler, Intel MKL, IB interconnect

intel_intel_1G → Intel compiler, Intel MKL, 1G interconnect

gnu_intel_1G → GNU compilers, Intel MKL, 1G interconnect

gnu_goto_1G → GNU compilers, GotoBLAS2, 1G interconnect

The vertical axis represents the performance of HPL measured in GFLOPS in each of the above configurations.

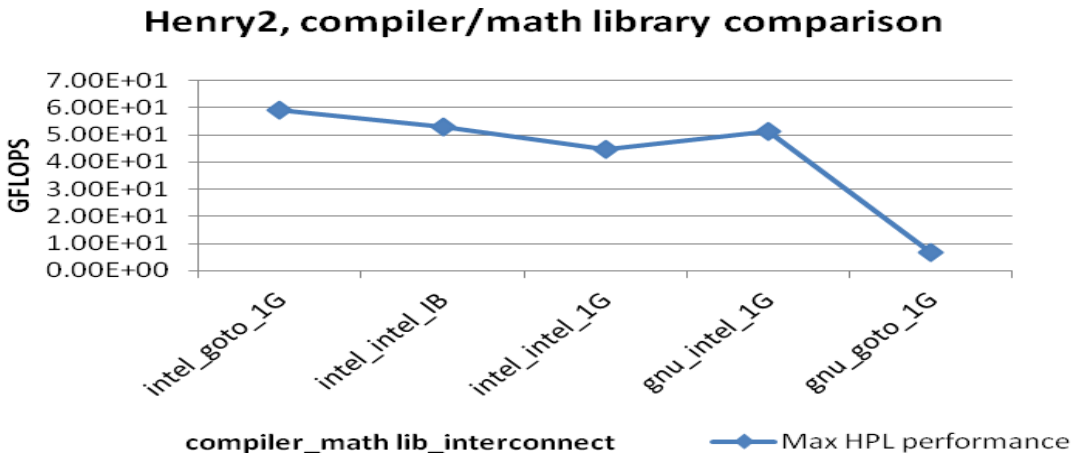

Figure 38: Comparison of different compilers and MKL libraries

III. A simple analysis of Ping Pong benchmark results:

A simple Ping Pong test was done with one MPI process on each node. To test the network interconnects with Ping Pong test, we have to execute one MPI process per node. This test is different from the above tests with multiple MPI processes per node where there was some opportunity for shared memory. We were successful to get the test done with 8 nodes and 4 nodes with a 1 Gb/s interconnect and 8 nodes with InfiniBand interconnect. For Figures 39 and 40, the horizontal axis represents different Ping Pong tests and left hand vertical axis represent latency for 1G interconnects and right hand vertical axis represent the latency for the InfiniBand interconnect in milliseconds. As expected, InfiniBand (denoted by "(IB)" in the figure) latency was observed to be lower that the 1G Ethernet. The Ping Pong latency as shown in Figures 39 (8 Byte message latency) and Figure 40 (2 MB message latency) was

observed to be consistent except for a spike in latency during one of the tests (the spike is around 25 ms and is an outlier). The graph in Figure 39 shows slight increase in latency as we increase the number of processes from 4 to 8. This is expected since as the number of processes increases more communication messages are sent across and hence more delay.

The bandwidth test is shown in Figure 41 for 1G Ethernet and 40 Gb/s InfiniBand interconnects (horizontal axis represent different ping pong tests and left vertical axis represent the network bandwidth in MB/s for 1G interconnect and the right vertical axis represent the network bandwidth for InfiniBand interconnect). Figure 41 depicts that as we increase the number of processes, the achieved bandwidth decreases slightly.

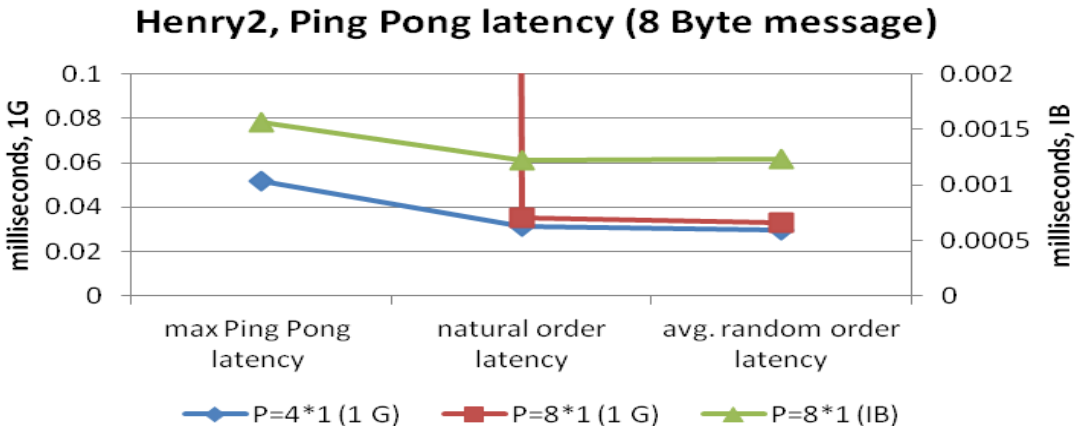

Figure 39: Ping Pong Latency, 8 Byte message

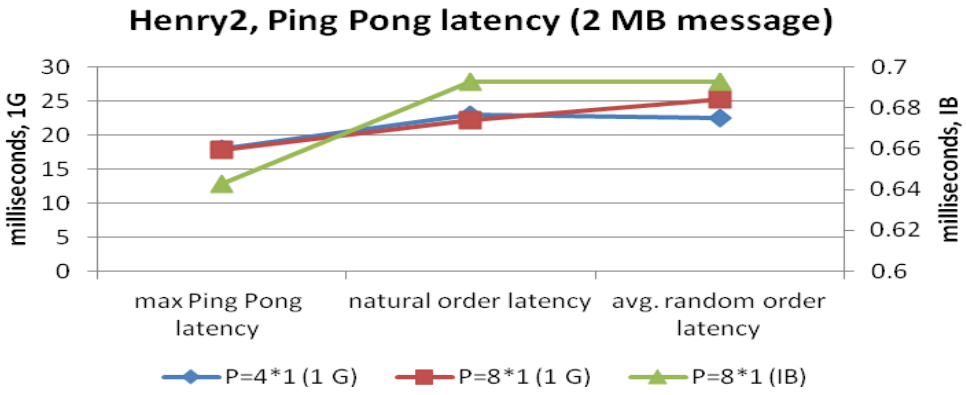

Figure 40: Ping Pong Latency, 2 MB message

Figure 42 is an interesting graph. It shows the relation between performance and minimum bandwidth achieved. The left vertical axis is a logarithmic scale to represent the minimum bandwidth achieved in MB/s and the right hand scale represents the HPL performance in GFLOPS. We get a slight increase in performance when we use an InfiniBand interconnect as compared to the 1G interconnect. But this increase might not be not be exclusively due to the InfiniBand interconnect. The CPU configurations for the two tests were different (1G test was executed on CPU configuration 6 while IB test was executed on CPU configuration 5). This is controlled by the LSF scheduler. We could not get the same CPU configurations for both the tests. Table 13 shows the min/avg/max latency and bandwidth achieved during these tests.

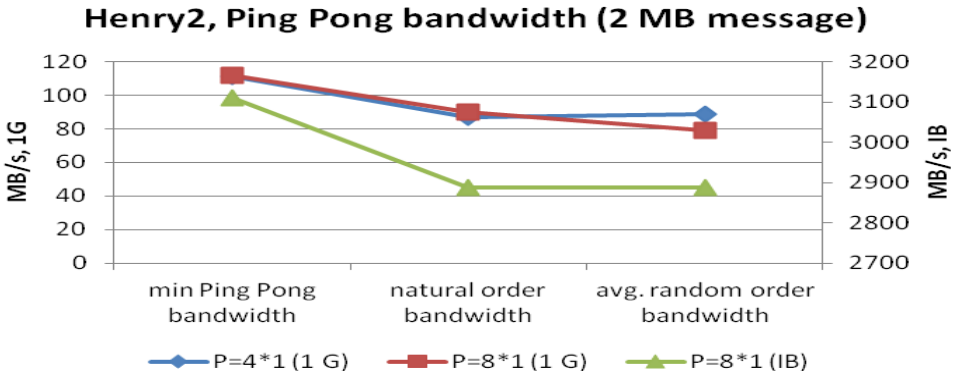

Figure 41: Ping Pong Bandwidth for 1G and InfiniBand interconnects (2MB message)

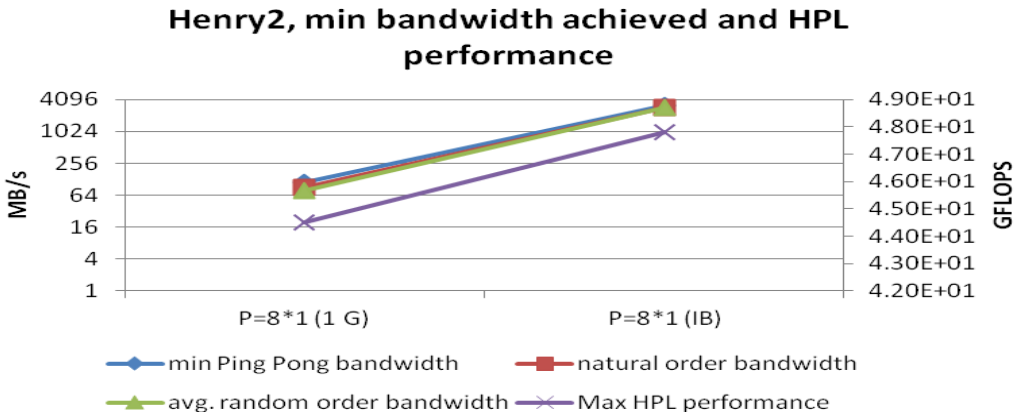

Figure 42: Ping Pong minimum bandwidth achieved for 1G and InfiniBand interconnects and the corresponding HPL performance

Table 13: Ping Pong Min/Avg/Max bandwidth and latency (Henry2 cluster)

Process configuration	2 MB message bandwidth (MB/s)			8 Byte Latency (milliseconds)			2 MB latency (milliseconds)		
	min	avg	max	Min	avg	max	min	Avg	max
P=4*1 (1 G)	111.176	115.657	116.492	0.042886	0.046951	0.051692	17.16852	17.29538	17.98952
P=8*1 (1 G)	111.835	116.116	116.598	0.040248	5.162538	25.03532	17.15303	17.22527	17.88354
P=8*1 (IB)	3110.42	3129.649	3149.61	0.001525	0.001539	0.001562	0.635	0.639054	0.643

IV. Max Performance:

We executed another interesting job to perform a maximum configuration test on the Henry2 nodes. We could get a maximum of 64 cores form the Henry2 cluster with the CPU configuration 5. This is a limit imposed by the LSF scheduler. We could test for higher configurations with special permissions but we wanted to do the test from a normal user's perspective. The results are as shown in Figure 43 (the horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS). We could reach a maximum of 392 GFLOPS with 64 cores. Assuming a balanced distribution of load, we get 6 GFLOPS per core (CPU configuration 5). The next graph in Figure 44 (the horizontal axis represent varying blocksize and vertical axis represents performance in GFLOPS) shows the HPL performance on 64 cores on the ARC cluster. We got a maximum of 329 GFLOPS on 64 cores on the ARC cluster which is 5 FLOPS per core. The difference between these two tests is that LSF scheduler on the Henry2 cluster allows for resource request at core granularity while the Maui scheduler on the ARC cluster has a per node granularity. Due to this, HPL spawned less MPI processes per core on the ARC cluster but is highly threaded.

A single MPI process is able to use all the cores available on a node of the ARC cluster because of the multithreaded GotoBLAS2 library. By default the OMP_NUM_THREADS is set to the number of cores in the node. All these parameters can be changed, but we preferred to test the clusters with default parameters since we want to answer the question of Hybrid Cloud computing which will use the in-house clusters with default parameters.

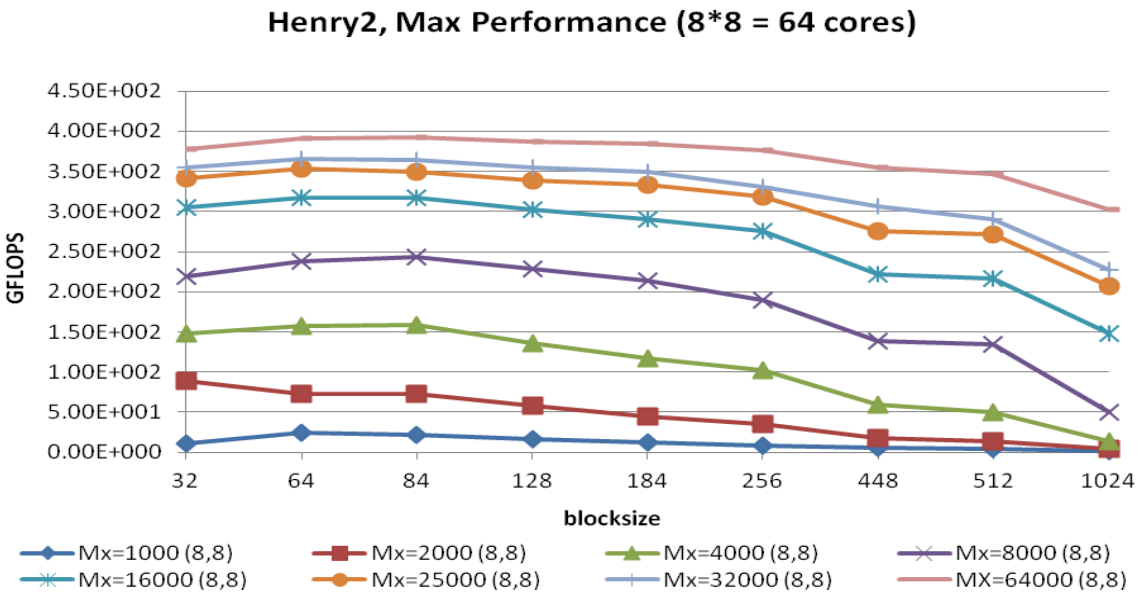

Figure 43: Max Performance for Henry2 cluster (8 nodes, 64 cores, Intel compiler/Intel MKL/IB interconnect, OMP_NUM_THREADS=1, CPU configuration 5)

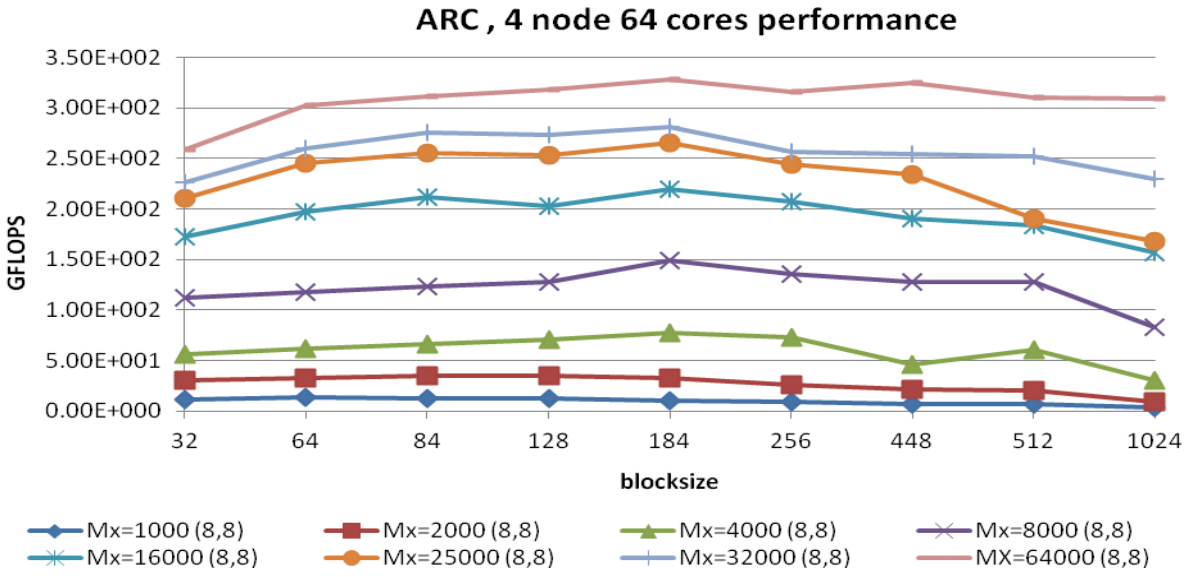

Figure 44: Performance for ARC cluster (4 nodes, 64 cores, OMP_NUM_THREADS=16, CPU configuration 2)

The outcome of these tests is that, these clusters, when compared to each other differ in several aspects of configuration as well as performance. Hence it can be concluded that a

user has to be aware of the underlying architecture and configuration if the user wishes to get maximum performance. An important point to make is that, to enable Hybrid Cloud Computing, the HPC cluster in cloud needs to be flexible to allow the user to change all those parameters that affect the application performance.

3.5 Flops performance results

Flops, is an old and a single threaded tool used to measure performance of a single core. The performance of a few CPU configurations mentioned above is tested using the Flops tool. We used four different compiler optimizations, -O4 (executable name=flops4), -O3 (executable name=flops3), register operands (executable name=flops2) and tests were also performed with no optimizations (executable name=flops). Table 14 shows the details of the processors we used in our tests.

Table 14: Processor configurations

Cluster	manufacturer	CPU	microarchitecture	clock frequency
Henry2	Intel	Xeon L5335	Core	2 GHz
Henry2	Intel	Xeon E5520	Nehalem-EP	2.27GHz
Henry2	Intel	Xeon E5649	Westmere-EP	2.53 GHz
VCL-GP cluster	Intel	Xeon L5638	Westmere-EP	2 GHz
ARC	AMD	Opteron 6128	Magny Cours (K10)	2 GHz

For each configuration an empty loop was timed and calculated. As expected with compiler optimizations the empty loop timer was zero. If compiler optimizations were removed, the empty loop timers varied according to the processor frequency. As expected, again, processors with higher frequency have low empty loop timers. In Figure 45, the horizontal axis represents the different compiler optimizations and the vertical axis represents the time required for empty loop in milliseconds.

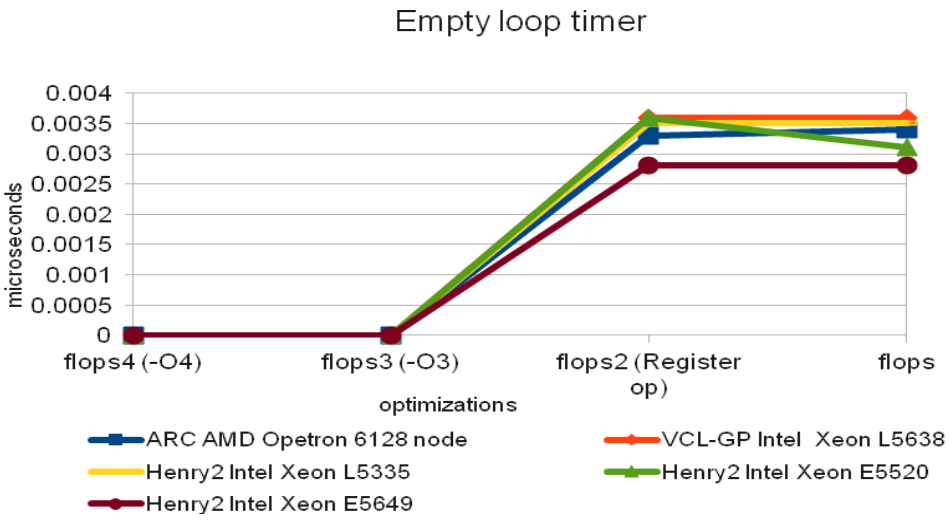

Figure 45: Empty loop timer

Following four different computation functions using double precision arithmetic are tested by Flops.

- MFLOPS (1) - FADD- 40.4%, FSUB 23.1%, FMUL-26.9%, FDIV-9.6%
- MFLOPS (2) - FADD-38.2%, FSUB-9.2%, FMUL-43.4%, FDIV-9.2%
- MFLOPS (3) - FADD-42.9%, FSUB-3.4%, FMUL-50.7%, FDIV-3.4%
- MFLOPS (4) - FADD-42.9%, FSUB-2.2%, FMUL-54.9%, FDIV-0.0%

It is observed that MFLOPS (4) test gives max performance on all configurations. Flops do not take advantage of processor pipelining or temporal/spatial locality of data offered by the cache architecture. Hence the performance of Flops is considerably lower than the theoretical limit. Yet it is representative of relative performance of each core.

I. Flops on CPU Configuration 1:

The maximum performance observed for a node in VCL-GP is 1718 MFLOPS using the Flops benchmark. As expected with better compiler optimizations we get better performance. In Figure 46, the horizontal axis represents the different compiler optimizations and the vertical axis represents the performance in MFLOPS.

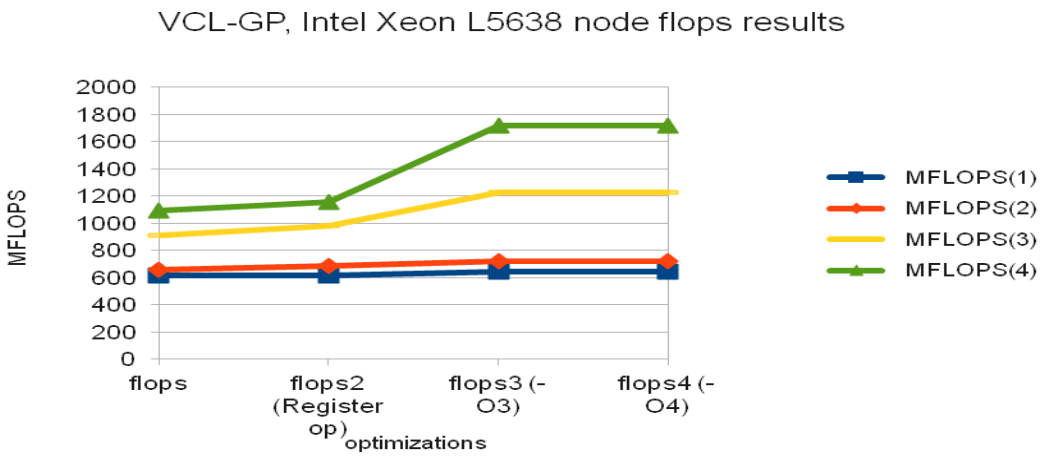

Figure 46: Flops results for VCL-GP cluster node (configuration 1)

II. Flops on CPU Configuration 2:

The maximum performance observed for a node with AMD Opteron 6128 CPU is 1359 MFLOPS using the Flops benchmark. As expected with better compiler optimizations we get better performance. In Figure 47, the horizontal axis represents the different compiler optimizations and the vertical axis represents the performance in MFLOPS.

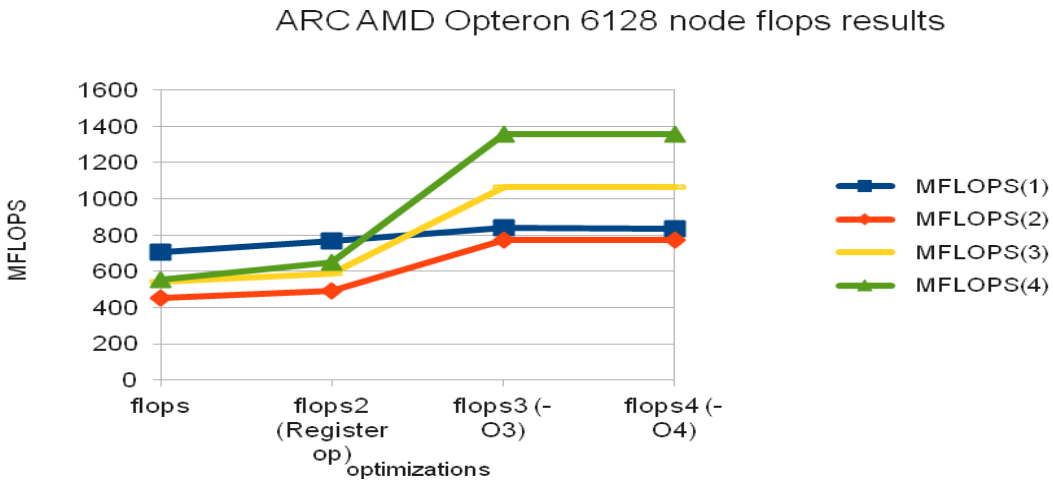

Figure 47: Flops results for Henry2 cluster node (configuration 2)

III. Flops on CPU Configuration 3:

The maximum performance observed for a node with Xeon L5335 CPU is 1333 MFLOPS using the Flops benchmark. As expected with better compiler optimizations we get better performance. In Figure 48, the horizontal axis represents the different compiler optimizations and the vertical axis represents the performance in MFLOPS.

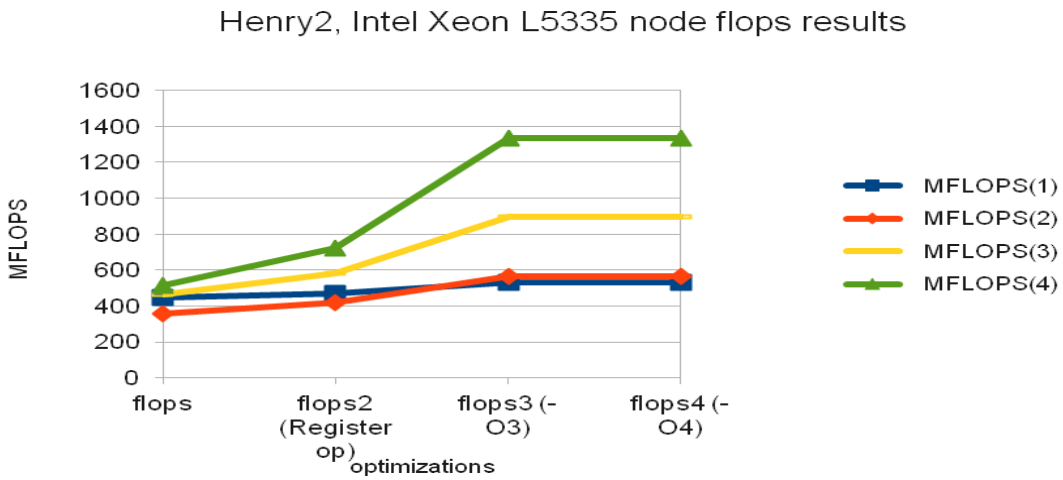

Figure 48: Flops results for Henry2 cluster node (configuration 3)

IV. Flops on CPU Configuration 5:

The maximum performance observed for a node with Xeon E5649 CPU is 1771 MFLOPS using the Flops benchmark. As expected with better compiler optimizations we get better performance. In Figure 50, the horizontal axis represents the different compiler optimizations and the vertical axis represents the performance in MFLOPS.

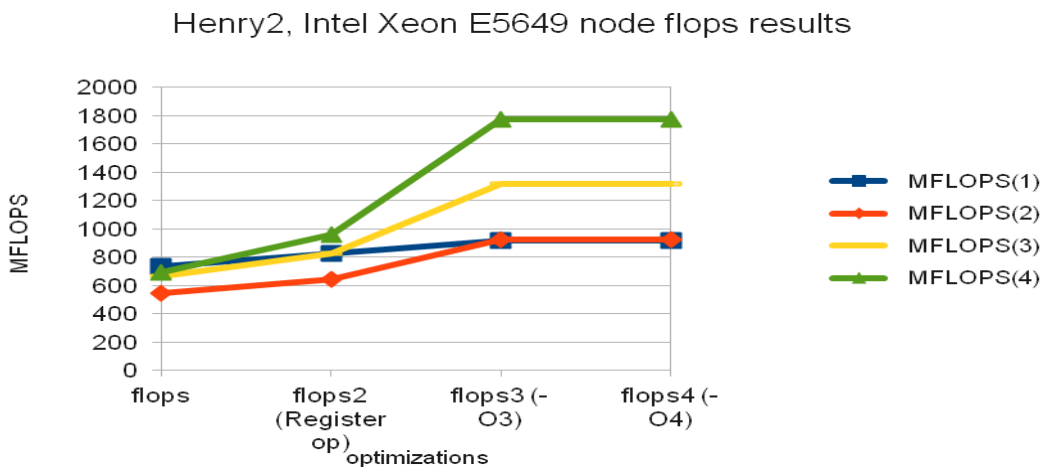

Figure 49: Flops results for Henry2 cluster node (configuration 5)

V. Flops on CPU Configuration 6:

The maximum performance observed for a node with Xeon E5520 CPU is 1585 MFLOPS using the Flops benchmark. Again, as expected with better compiler optimizations we get better performance. In Figure 49, the horizontal axis represents the different compiler optimizations and the vertical axis represents the performance in MFLOPS.

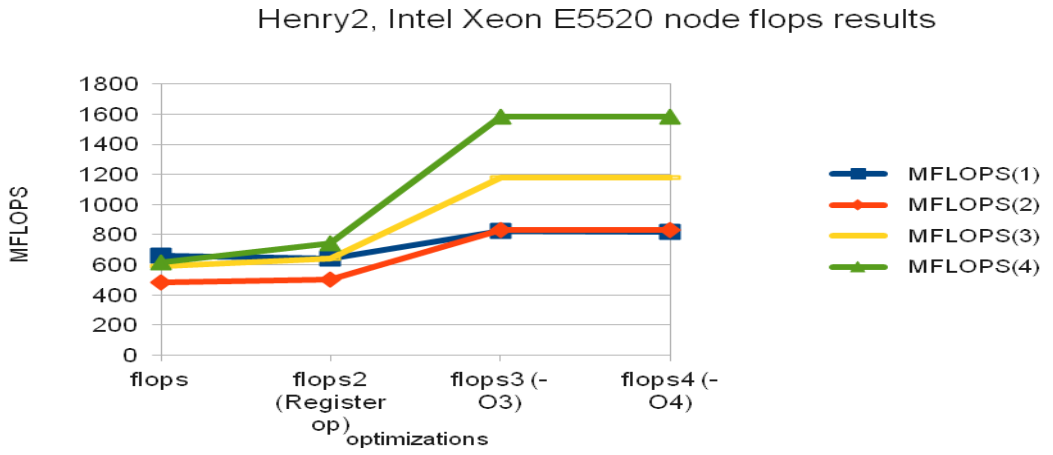

Figure 50: Flops results for Henry2 cluster node (configuration 6)

The observations in the graphs above confirm that with newer CPU's form the vendor in addition to greater clock frequency of operation, we get better Flops performance. The Flops

benchmark is a simple C program performing a combination of floating point operations. The test might not be able to achieve the peak maximum since Flops is not able to fill the pipeline with continuous instructions. We have tried further compiler optimizations like "-funroll-loops" "-fomit-frame-pointer" but the results were similar to a "-O3" or "-O4" optimizations. Note that Flops does not operate on large amount of data so cache optimizations are ignored here. A further optimization of using register file for storing the floats is used in the operation. The result is shown as column "flops(Register op)" in each graph. Using the register file gives a better performance but it is not significant enough to make a statement and stand out.

The experiment although not exploiting peak performance as HPL, shows that single thread performance for the various CPU's is consistent and does not depend on the fact that the CPU is a cloud resource or an in-house cluster resource. This experiment makes sure that VCL does not affect the performance of a single core standalone processor.

3.6 Deep study of the Ping Pong Latency and Bandwidth test on VCL-GP

We have seen in section 3.2 that the performance was related to the maximum latency and minimum bandwidth achieved by a Ping Pong test. We investigated the logs further to understand the maximum latency observed for 2 MB messages and the correlation to the performance of HPL. The following three graphs in Figure 51, 52 and 53 explain the results observed for VCL-GP at four different times T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week. In Figure 51 and 52, the max latency maps to the left hand scale while the performance maps to the right hand scale. Similarly in Figure 53, minimum bandwidth maps to the left hand scale and performance maps to the right hand scale.

Figure 51 illustrates that the HPL performance increases with decrease in max latency for an 8 Byte message. We can argue that the bandwidth utilization is high when 2,000,000 Byte messages are sent across. Hence it is worth to note the latency for a 2MB message. Figure 52, shows that the maximum latency for an 2 MB message at time T1 was around 223

milliseconds and for T2, T3 and T4 (T4 = T3 + a week) it was 200, 32 and 32 milliseconds respectively. The drop in max latency is reflected in the performance of HPL.

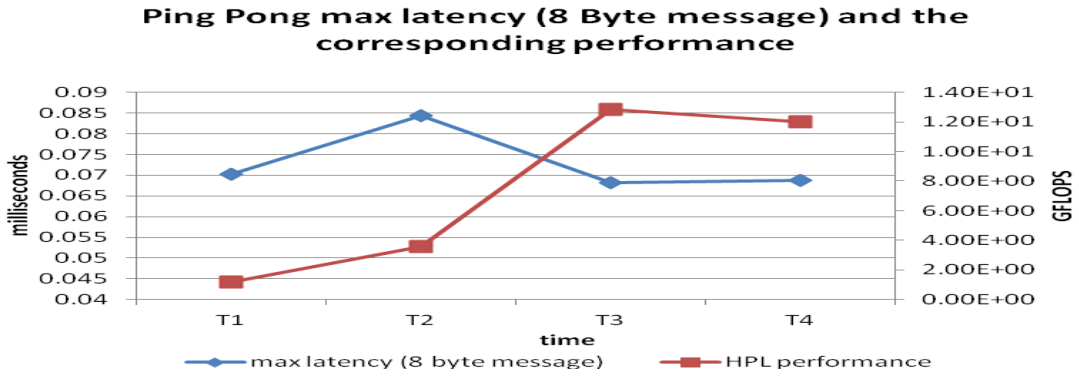

Figure 51: Max Ping Pong latency for 8 Byte message and the corresponding HPL performance (4 node, 4 core, T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week, VCL-GP cluster)

This observation is in perfect harmony with the HPL performance numbers. The maximum latency at a particular time and hence the minimum bandwidth achieved at this time seems to be one of the reasons for the variability in performance seen at that particular time.

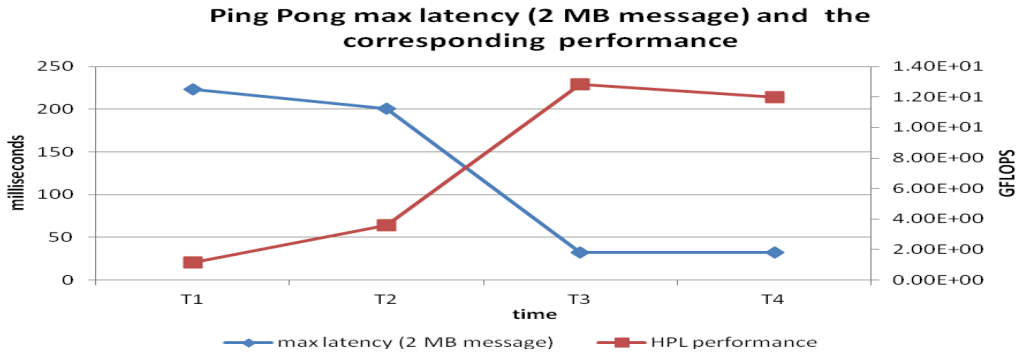

Figure 52: Max Ping Pong latency for 2 MB message and the corresponding HPL performance (4 node, 4 core, T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week, VCL-GP cluster)

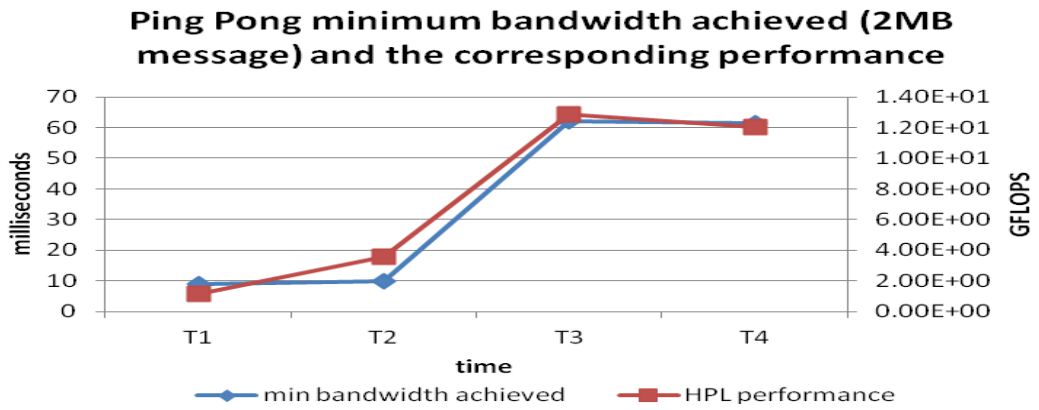

Figure 53: Min Ping Pong bandwidth achieved for 2 MB message and the corresponding HPL performance (4 node, 4 core, T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week, VCL-GP cluster)

Figure 53 reflects similar results. As the minimum bandwidth achieved increases, the HPL performance gets better and better. We cannot identify a particular cause for the variability in the latency or bandwidth achieved at different times, although additional load on the involved switches is a natural suspect. Table 15, shows the minimum, average and maximum latency and bandwidth achieved at T1, T2, T3 and T4.

However, these observations illustrate that, in the worst case scenario, a general purpose cloud network performance may have large variability and that may introduce variability in the cluster performances. Performance stability of Amazon EC2 clusters was studied in [27] and the observations suggest that the performance of HPC in cloud is variable, which is similar to what we see in VCL.

To achieve performance stability we need to create specialized zones or clusters of high-end nodes connected via high speed interconnects (InfiniBand, 10G Ethernet, etc). This means we would need to create in-house HPC clusters within a cloud subsystem. Various options for enabling such services or extending the capability of in-house clusters via cloud services are explored in the later sections.

Table 15: Minimum, Average and Maximum latency and bandwidth reported by Ping Pong test at different times (T1, T2 → T1+2 days and T3→ T2 + 2 hours, T4 → T3 + a week) VCL-GP cluster, 4 node, 4 core test

	2 MB message bandwidth (MB/s)			Latency (8 Byte message) (milliseconds)			Latency (2 MB message)		
Time	min	avg	max	min	avg	max	min	avg	max
T1	8.958	19.908	32.532	0.057069	0.065509	0.070315	61.478441	112.7334	223.2619
T2	9.952	19.152	41.937	0.055818	0.067725	0.084314	47.690985	118.8662	200.9695
T3	62.167	63.109	64.249	0.055753	0.062827	0.068119	31.12	31.69	32.17
T4	61.385	63.164	65.025	0.056631	0.063707	0.068745	30.757524	31.67637	32.58149

Chapter 4

Hybrid Compute Clusters in VCL

The performance characterization in Chapter 3 provides an interesting perspective of HPC in a cloud. Basically, unless one has a specially constructed pre-configured cluster and even then there is a distinct possibility that different cluster/cloud components may exhibit large variability in their performance. So, although HPC in a cloud is feasible (HPC-VCL works well), compute intensive applications may need to be tightly bound to the low level architectural traits of a particular cluster. The factors that affect the performance of an application are essentially the processor compute power (including parallelism capabilities), memory capacity, memory bandwidth and latency, the network interconnect between the compute nodes, and in the case of I/O intensive applications the I/O bandwidth and capacity (i.e., the file system).

From the Chapter 3 discussions it is clear that pure (traditional) HPC in a cloud (VCL-GP) may be a challenge if the only available resources are general cloud nodes. Node networking topologies, as well as other characteristics need to be controlled, even in the case of embarrassingly parallel applications. This necessarily does not mean one cannot implement HPC in a cloud, but it does open new avenues of research to engineer new options with regards to enabling the HPC compute services through a cloud subsystem.

This chapter discusses in more details some ways of engineering HPC and possibly high-performance data (HPD) services in a cloud subsystem. One method is re-direction of user workflows to specialized pre-configured clusters. Redirection is useful in the following use cases:

- Scientific workflows are designed to execute a series of computational steps. One of the steps might include setting up a secure channel to a remote or local HPC subsystem and execute the jobs. A redirection technique is used to set up to channel such requests to the HPC system.

- As we have discussed in chapter 1, the elastic Nimbus project [50] if integrated with VCL, would require a secure channel inside the VCL subsystem to communicate with the VCL management node. This can be achieved by using a redirection server which can forward requests from a source resource manager to a destination resource manager.
- In chapter 5, we discuss the possibility of VCL integration with Kitty-Hawk, which enables executing native Linux on BlueGene/P [92, 93] architecture. To enable the communication between Kitty-Hawk and VCL, we have suggested the use of a redirection technique.

The following section explains the different ways of engineering redirection techniques using a redirection server we call a gateway server.

4.1 Redirection

In VCL one can construct and integrate services in different ways. One is to offer service components to VCL through a local or remote provisioning module and have VCL configure, control and use them, another is to construct "gateways" using existing resources (e.g., through Linux or Windows-based applications) tunnel to the requested services. Latter usually means that the resources are used in a differentiated mode.

For example, VCL already provides VCL-HPC access nodes, and as a part of this work, we have put in place a trusted gateway node for access to ARC - a custom built GPGPU cluster. This loosely coupled solution provides flexibility to the users to choose in their workflows, the kind of service that they need. If someone needs a VCL-HPC cluster, they can be redirected to the VCL-HPC cluster, or if a user requires GPGPU clusters with CUDA capability, the user can be redirected to the ARC cluster.

The basic idea of a trusted gateway is to securely redirect requests to requested services. Figure 54 illustrates this. There are several ways in which a VCL node can redirect the users to the VCL-HPC, to the ARC cluster or to the head node of another customized cluster b.

One is an ssh tunnel using keybased authentication, and either direct (e.g., via tunnel) or indirect (e.g., via NFS or other type of network attached storage) transfer of data to and from the target cluster. . In this work, we have designed and implemented a redirection gateway to work with the ARC cluster.

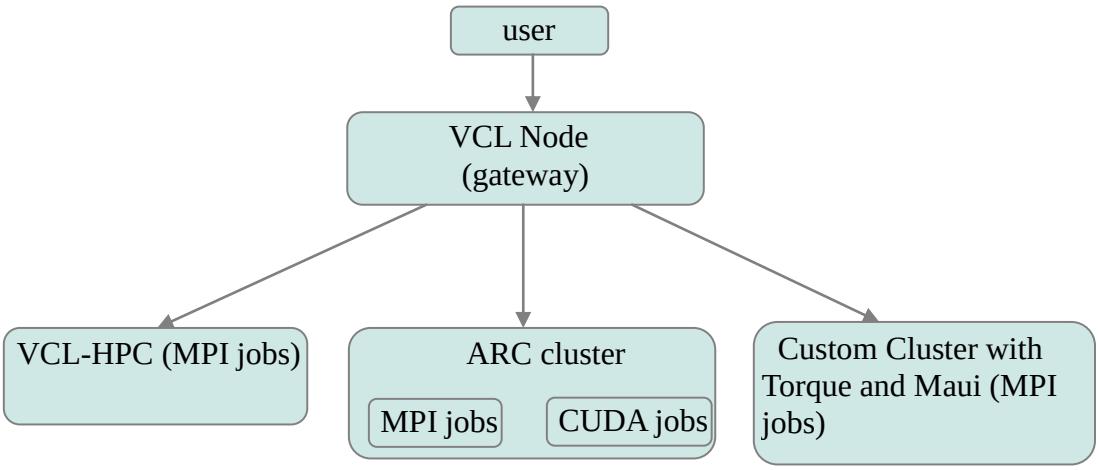

Figure 54: High Level VCL gateway provisioning architecture

4.2 Implementation

We add a new layer of abstraction to the existing infrastructure. Following two methods work hand in hand to achieve the goal.

4.2.1 Accessing ARC cluster via VCL

We use a VCL node as a forwarding agent or a gateway. Figure 54 provides a good pictorial understanding of the scenario. For this to be realized on linux (via ssh interactive shell) we need to redirect the ssh connection into the gateway onto the destination. There are various ways to do so.

SSH redirection using

i. ~/.bashrc:

When a user logs onto the server, the ssh server spawns a new bash shell. The bash default rules for the user are read from the file /home/$USER/.bashrc. A system administrator can force commands to a bash shell. Hence to forward the ssh login session, add the following line to the .bashrc file.

“ssh -l $USER <destination ip>”

In our scenario, we add the following line, “ssh -l $USER arc.csc.ncsu.edu “.

ii. /etc/bashrc:

Although the above method works, it does not work for all users. It will work for $USER and in VCL $USER cannot be constant. For each user VCL creates a new user account and each use has its own home directory. Hence the ~/.bashrc does not work in this scenario. We can always write a bootup script which modifies the ~/.bashrc for that particular user but there is an easier way. We add the above redirection command to /etc/bashrc which is read on invocation of all bash shells by any user. This is a more general approach.

iii. /etc/profile:

/etc/bashrc is a common configuration file for bash shells. What if some user prefers a bourne shell or a tcsh shell or some other shell? In addition, not all unix flavors have a bash shell. Some restrictive shells can also be spawned. The “/etc/profile” configuration file is read by all shells. If we add the above redirection/forwarding command to /etc/profile, it is executed by all shells. This is a more general approach than above.

iv. ~/.ssh/authorized_keys:

It is clear that we have covered all shells and if any shell is spawned, we will be redirected. Now what if, a user does not require a shell? What if a user just sends ssh commands without a 'tty' required? None of the above configuration files will be executed and the commands will execute on the middle gateway server. To avoid this we redirect/forward ssh traffic by forcing the commands to the destination server. We can force commands to any ssh connection. This is done by adding commands to the authorized_keys file. The general syntax to force ssh commands is:

command=”< forced command>” ..... user key.....

Each command is bound to a particular user key. If a connection is incoming from that particular user, ssh forces the execution of the command in the *authorized_keys* file.

To understand what command we should force for redirection, we need to understand how the ssh command works. Let’s take two examples of a remote command invocation via ssh and the scp command.

Eg 1: Simple ssh command forwarding:

Suppose we have three machines as shown in the Figure 55,

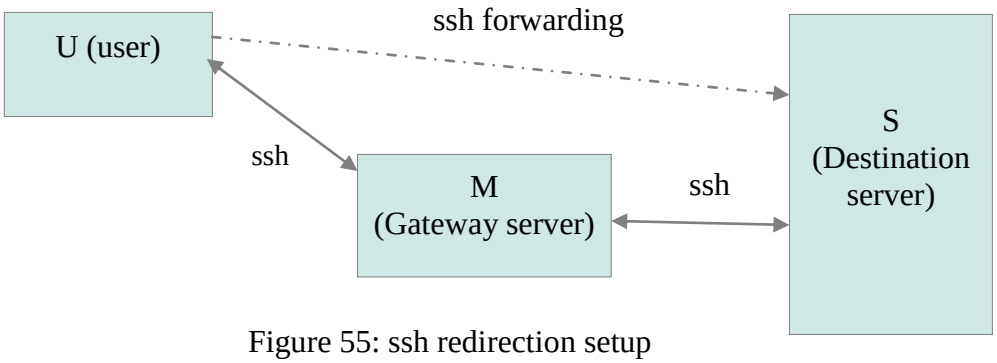

Figure 55: ssh redirection setup

The goal is to redirect all ssh and remote command invocations using ssh that are incoming from U towards M. The destination is server S. Suppose we fire a remote ssh command from U, *ssh -l $USER M <command>*, we would expect the <command> to run on M and return the output of *stdout* back to U's *stdout*. But here we do not want the command to run on M but on S. For this we have setup ssh forwarding by including the following line in the *authorized_keys* file.

command=”bash -c 'ssh -l $USER S ${SSH_ORIGINAL_COMMAND:-}'” … user-key …..

This will force the command to execute and forward the ssh command to S. To make this possible, the ssh server saves the incoming ssh command in an environment variable called *$SSH_ORIGINAL_COMMAND*. If we fire *ssh -l $USER S uname -a* then *uname -a* is saved

in *$SSH_ORIGINAL_COMMAND*. The *${SSH_ORIGINAL_COMMAND:-}* is required. It works in the following way, if ${A:-B} then if $A is not empty, print string B. Here we keep B empty since we do not want this forced command to mess with the actual ssh command which was fired to access the interactive shell prompt [16, 19]. Deep explanation of how secure shell works can be found in [16], eg 2: *scp* command forwarding. For this we need to understand how the simple *scp* command works.

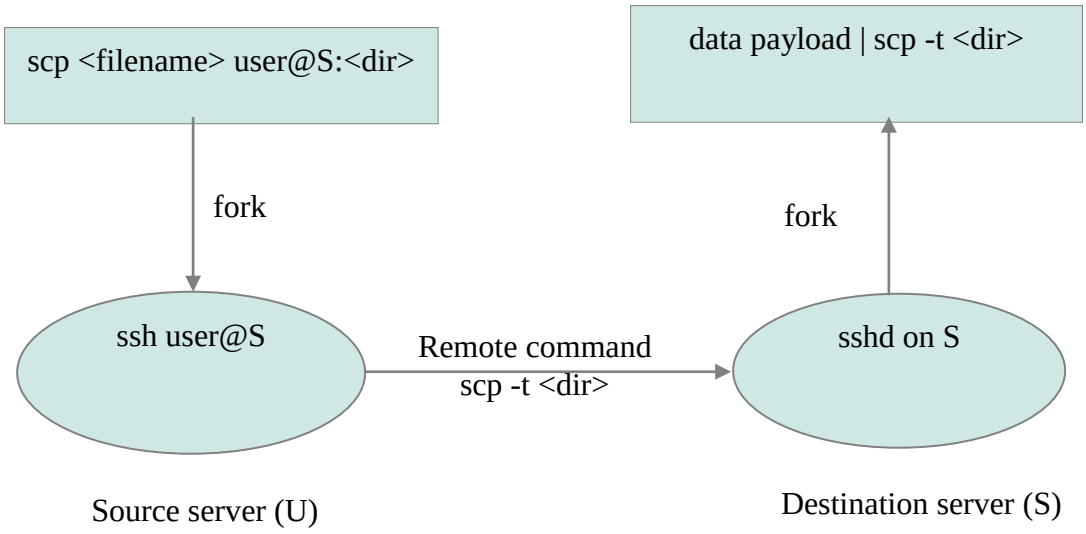

Figure 56: How *scp* works [19]

As shown in Figure 56 above, the scp client forks and new ssh connection to server S. the ssh client on U or source establishes a ssh connection to S. It sends a remote ssh command *scp -t <destination dir>* which runs on S as a local command. The input to scp server side program is a data payload which is identified and parsed by the scp server on S and the data is written to file as specified in the data payload. The scp works in either source mode or sink mode. In source mode (sender) it sends simple text messages to destination and runs a remote command to generate the file. In sink mode (receiver), the above messages from source are acknowledged accordingly. The source sends the filename and other protocol messages and then sends a zero byte to notify the receiver that the protocol messages are over and now the

data begins[16, 19]. More and in depth information can be obtained from [16] and [19]. Let us take and example to explain this in detail.

For example: *{ echo C0644 6 hello; echo hi there; } | scp -t /home/user*
This will generate a file called *hello* in */home/user* with data *"hi the". C0644 6 hello*, is the scp protocol for sending filename and attributes. 6 is the length of the file in bytes. Scp server on S will copy 6 bytes from the next incoming payload data to the file */home/user/hello*. Now instead of 6 if we say the length is 8, the contents of the file would be *"hi there".* This is how scp works [16, 19].

Back to where we began, if we want to forward the scp data via M to S, as shown in Figure 55, we need to redirect the scp command by changing the *authorized_keys* file. The ssh command,

command="bash -c 'ssh -l $USER S ${SSH_ORIGINAL_COMMAND:-}'" … user-key …..

works for scp too. All it does is sends the remote command *scp -t <dir>* to S. Thus scp forwarding works. But there are also issues with this technique. Although, this is the best way for forwarding, it can be applied per user per key. For this to work transparently, every user must be authenticated to M using a key-based authentication rather than password based. So we need to get each user to upload a public key to M. In our scenario this fits in rather perfect. The ARC cluster authenticates users based on their keys. So the user needs to upload a public key to the ARC cluster. The same key can be uploaded to the server M which is nothing but a VCL image. It is also possible to get the key for that particular user form ARC cluster itself if it permits, but we have not explored this option since we wanted the design to be completely independent of the ACR cluster. Another issue is that this works per user. To make this work for all users, a script needs to be written which will generate such commands and append them to the *authorized_keys* file.

We can successfully redirect the session to the remote machine, but the question remains, how to provide key based authentication to the VCL image 'M', as shown in Figure 55. VCL uses the usual NCSU wrap (shibboleth) authentication. Currently a user does not need to upload keys to the VCL node to access the node. There needs to be a mechanism where the user needs to upload the keys to the VCL node in order to access the middle machine M and in turn get access to the ARC cluster. To provide for a mechanism for uploading a user specified ssh public key to the server M, a web server runs on the server M which can be accessed by typing the following link in any web browser

<IP address of M>/myupload/upload_file.html

We know the IP address of M form the VCL reservation page. This web-server provides for a simple file upload mechanism. The user needs to upload the ssh public key to this web server. Internally a script is running which identifies the key upload and updates the authorized_keys file for the user to get authenticated via ssh-key. The script also modifies the authorized_keys file to enable forced commands. In this scenario, the script updates the authorized_keys file with the redirection command as shown.

command=”bash -c 'ssh -l $USER S ${SSH_ORIGINAL_COMMAND:-}'” user-key ….

Thus, we can facilitate the forced commands as explained above and provide for an automated but secure redirection to the ARC cluster.

The web-server design is pictorially explained in Figure 57 below. Although the user requests a reservation using its NCSU unity id via the VCL website and VCL creates a user with the unity id of the user for authentication, the usual password authentication is disabled on this VCL node. This is done on purpose since the user does not need access to the middle server M as it is required to be transparent to the user. The purpose of M is to enable access to the final destination cluster (ARC cluster in our context). This is a security mechanism to protect against agent forwarding loopholes, explained later in the document.

In the future, it will be mentioned on the VCL page, in the reservation section, that the user cannot login to the server M via password authentication and hence should upload the ssh public key to the server M by accessing the web page,

"<IP address>/myupload/upload_file.html".

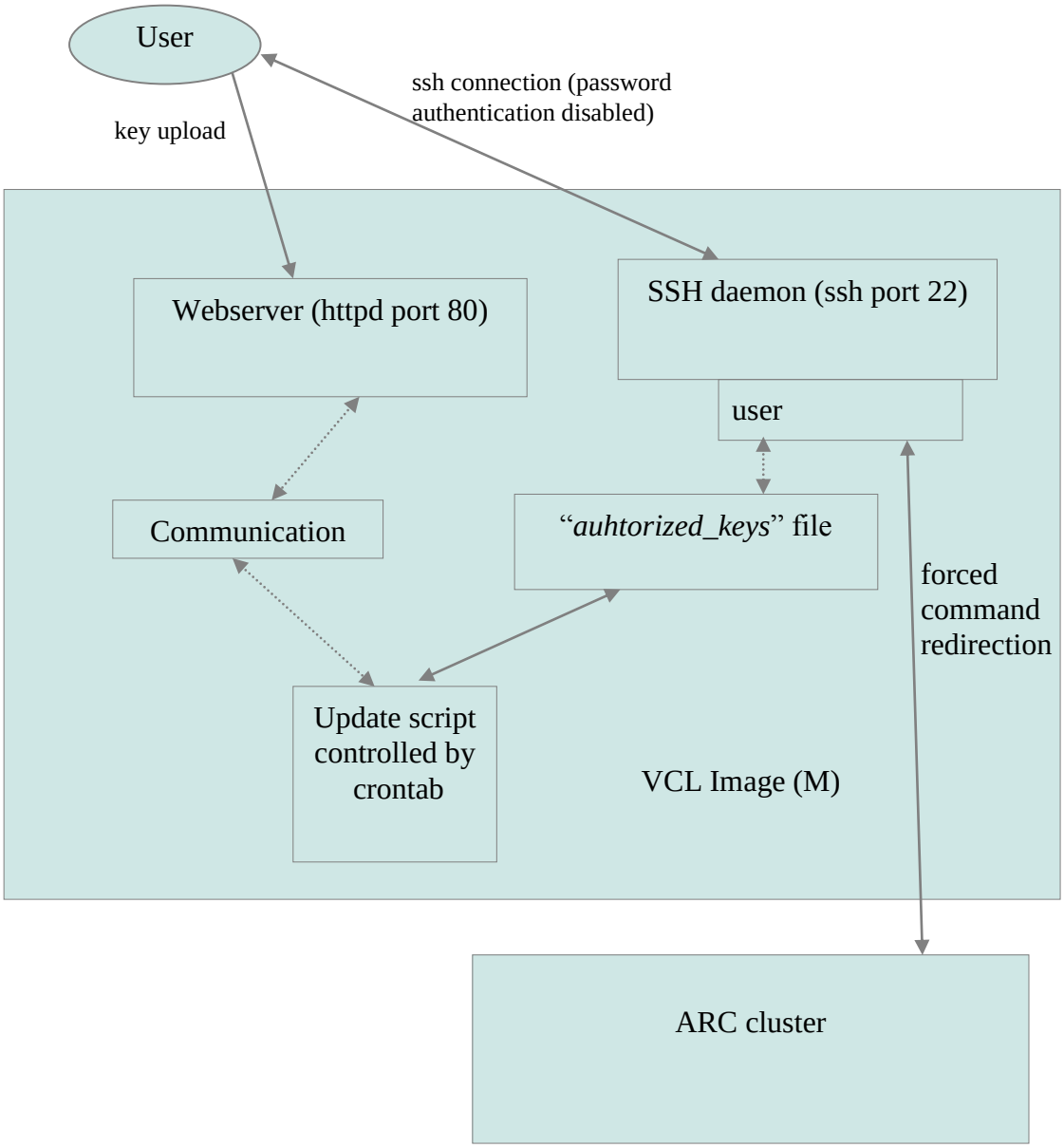

Figure 57: Web-server design

The web page will display an upload form as shown in the snapshot in Figure 58. The user has to generate a new ssh key pair using the 'ssh-keygen' command or can use the same public key that was uploaded to the ARC cluster during initial access. It is not necessary to use the same ssh public key uploaded to the ARC cluster. It is recommended that the user uses a new key for authentication to M. The decision is left at the discretion of the user.

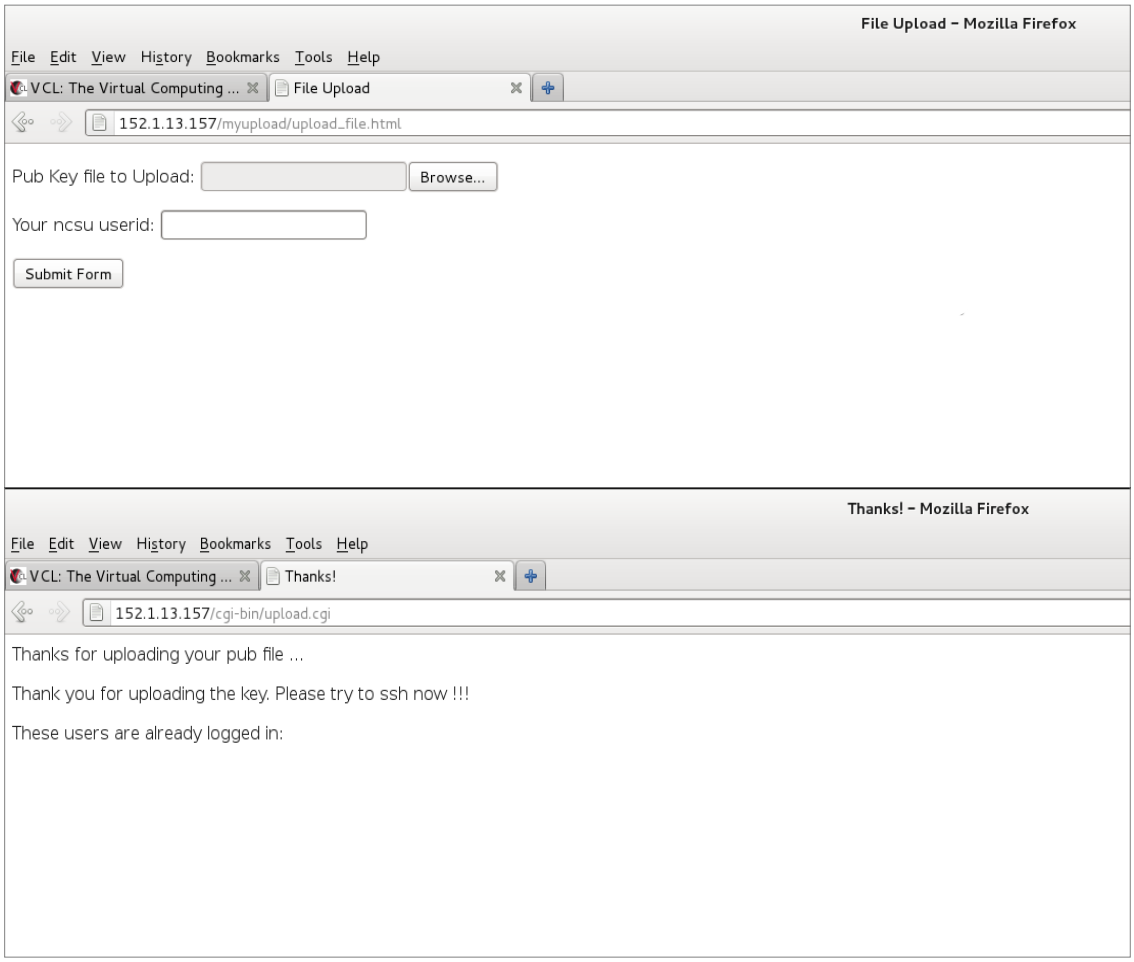

Figure 58: Web server snapshot

The file looks like this after the upload script finishes.

command=”bash -c 'ssh -l $USER S ${SSH_ORIGINAL_COMMAND:-}'”.....user-key

……

One could argue that this update could be done by the web server itself. But the issue is that the web server runs as user “unknown” and does not have permissions to access or update any files in the /home/<user unity id> directory hierarchy. Such an update is possible only if a script runs as root. This is the reason why we need the update script to run as a different entity and with root permissions. A script with root permissions can do a lot of damage to the server M. Hence the update script is written separately from the web server so as to protect any malicious activity. The web server can only talk to the update script via the communication file. As soon as the web server indicates the upload complete, the update script updates the *authorized_keys* file accordingly and shuts off the web server. The web server is no longer accessible after the upload. The update script also does some checks to ensure that only the user as indicated by the VCL reservation has permission to upload the keys. But these checks are very primitive and do not interface with the NCSU wrap authentication service.

As soon as the public key is uploaded to the web server, the web server writes to the communication file as shown in Figure 58, indicating the file update complete to the update script running in the background. The update script then copies the public key from the web server directory to the /home/<user unity id>/.ssh/authorized_keys file. Before the keys are updated to the authorized_keys file, the update script writes the forced command to the file.

Apart from the issues and the solutions discussed above, it is observed that that the scp command forwarding sometimes gives permission denied errors. This is because a forced command does not have an interactive shell to enter authentication information from M to S (Figure 55). For this to work, the ssh client on U needs to have an entry for *ForwardAgent yes* in the *ssh_config* file. The file is located at */etc/ssh/ssh_config*. This works as long as remote ssh commands are sent to the ARC cluster. The scp command fails even if the forwarding agent is enabled for the ssh connection.

After careful observations for constant errors in scp, it is observed that the scp command as opposed to the ssh command does not work with forwarding agent support. It is evident in

the debug trace provided in Appendix A.1 which correspond to the traces for ssh connection using agent forwarding, A.2 which show how traditional scp fails to use the agent-forwarding mechanism and the traces in A.3 portray how a modified scp with agent-forwarding implemented successfully copies the data over to the destination server.

As we can see that the trace A.1 clearly shows how the ssh client on U initiates the agent-forwarding. In trace A.2, U never initiates the agent-forwarding and hence the copy fails due to permission denied errors. Trace A.3 show the traces when agent-forwarding is forced on in scp. The traces show that the user U now initiates the agent-forwarding and the copy is successful. The trace shows that ssh uses agent forwarding but scp does not. This is because; the support for forwarding agent in scp is not implemented since this seems to be a security loophole. To understand the loophole better one needs to understand how the ssh agent forwarding works.

SSH Agent forwarding:
The man page for ssh says, "Agent forwarding should be enabled with caution. Users with the ability to bypass file permissions on the remote host (for the agent's UNIX-domain socket) can access the local agent through the forwarded connection. An attacker cannot obtain key material from the agent, however they can perform operations on the keys that enable them to authenticate using the identities loaded into the agent."

Let us first understand what this means. Figure 59, explains how ssh-agent forwarding works and how a malicious user can take advantage of it.

When a user U (Figure 59) tries to ssh to S (Figure 59), the following things happen:

- Since we are using M as a gateway, agent forwarding needs to be turned on. The user U requests agent forwarding when establishing the connection to M.

- ssh server on M starts a new agent to handle agent forwarding. The agent talks via the IPC communication channels i.e. Unix domain sockets and acts as a proxy agent for the connection to destination server S.
- Now we redirect the user U by starting a new connection to S.
- The ssh client on M needs key authentication to connect to S. The ssh client on M talks to the agent on M via unix domain sockets to request the key.
- The ssh server on M interprets this request, since it is acting a proxy agent, forwards the request to the original agent running on U via the ssh channel between U and M.
- The agent on U processes the request and sends the response to the proxy agent on M.
- The agent on M authenticates the connection from M to S and thus a ssh connection is established.

This mechanism works even if there are multiple such gateways in between U and S. More detailed explanation can be found in [16] and [102]. The security loophole here is the unix domain sockets. These sockets are implemented as simple files residing on the file-system with normal user permissions. A malicious user with user file permissions can take advantage of these sockets. A malicious user can read from these sockets since these are just files on a filesystem and thus utilize the keys that the forwarding agent is using. Thus if M is compromised, the user keys are accessible to all the users on M. In our scenario M lies in a trusted network (VCL) and no other user is allowed to login to M.

Even though the openssh developers have not implemented the ssh agent forwarding since it is considered as a security loophole, it is not a loophole in the scenario here. The system in the scenario here is protected by the NCSU shibboleth authentication. One has a protection of the shibboleth authentication as well as the key based authentication. The question of a third party malicious user taking advantage of the unix file permissions on M and thus compromising the key-based authentication has limited applicability in this scenario. But this

is still an open research topic and more scenarios can be explored to strengthen the provisioning module in the future.

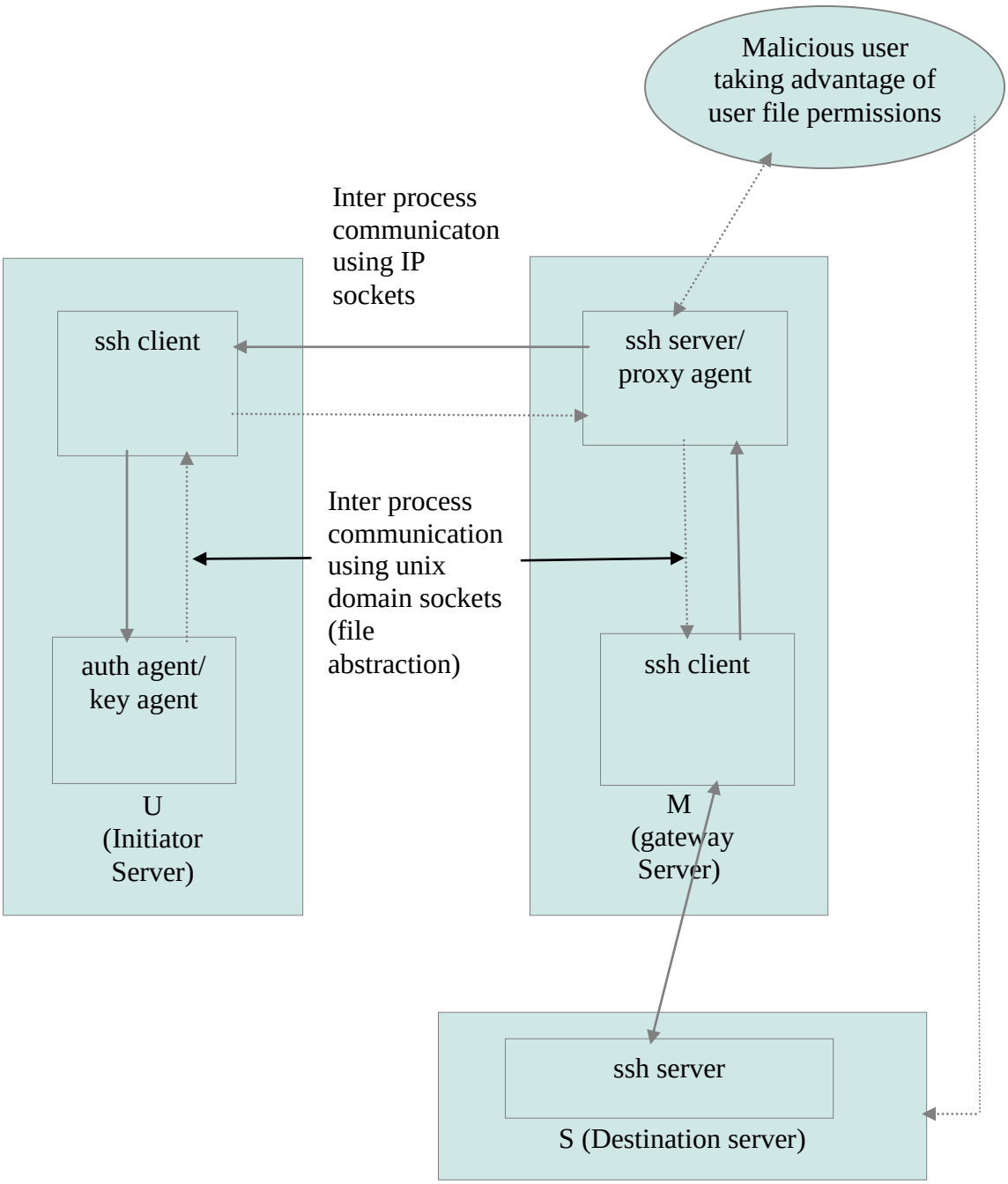

Figure 59: ssh agent-forwarding

Now that we understand the security loophole and concluding that it is not a threat in out scenario, we modified the scp code to enable ssh-auth agent in scp. This is a minor change and does not affect the scp functionality a lot. This just extends the capability of scp. The new code is compiled with user options which install the new scp in the location */opt/bin* and does not replace the preinstalled version of scp. The preinstalled scp is in */usr/bin/scp* and can be used for other secure copy purposes. We used the new binary for the scp to copy data to the final destination via the gateway we built. This provides for a complete flexibility for the VCL gateway node to move across the cloud subsystem. It does not depend on any NAS solutions to be implemented to provide complete abstraction. Now that we have complete abstraction ready, we tried to profile the newly built scp which copies data to the final destination via the gateway server.

Performance of scp without abstraction and with abstraction:

A simple scp test proved that there is delay introduced by the gateway (redirection) server. The results are noted in Table 16. The delay is not seen in large files. The latency is introduced when copying a directory with large number of files. As we can observe from the table, the latency introduced in copying a standalone bulky executable is negligible. But the latency introduced in a directory copy with large number of smaller source files is quite visible. This is because scp has to send all the file names along with the data in case of directory. Hence it is expected to take longer than copying a single bulky file. The delay because of the gateway server is quite high when a directory is copied.

Table 16: Performance of scp when redirection server is introduced

copy to	scp type	File	Type	size	copy time	copy rate
directly to arc	/usr/bin/scp	a.out	file	442 MB	195 s	2.26 MB/s
directly to arc	/usr/bin/scp	GotoBLAS2	dir	41.3 MB	62 s	0.667 MB/s
directly to arc	/opt/bin/scp	a.out	file	442 MB	205 s	2.15 MB/s
directly to arc	/opt/bin/scp	GotoBLAS2	dir	41.3 MB	65 s	0.635 MB/s
via vcl gateway	/opt/bin/scp	a.out	file	442 MB	215 s	2.05 MB/s
via vcl gateway	/opt/bin/scp	GotoBLAS2	dir	41.3 MB	118 s	0.35 MB/s
via vcl gateway	/usr/bin/scp	a.out	file	442 MB	n/a	n/a
via vcl gateway	/usr/bin/scp	GotoBLAS2	dir	41.3 MB	n/a	n/a

There were delays expected in this kind of a setup. These delays are introduced by TCP/IP stack processing on the gateway server. Since scp is a layer 5 protocol that works on top of the TCP/IP layer, the delays get added with each such redirection. Hence it is advisable to keep the redirection to bare minimum.

This delay can cause fatal when the submitted jobs perform a remote copy of user data during the compute operation. This is rarely observed since compute intensive applications are focused on computations on a pre-populated user data.

As the section helps to understand how the abstraction in VCL works, this VCL node can be used as an abstraction over any existing service in the cloud or any future services. A user requirement can change drastically and considering the discussion in Chapter 3, every other in-house cluster has different performance characteristics. Depending on the user requirement, we might need to provide access to the user to different clusters as the requirement changes. This should happen on the fly and should be transparent to the user. This requirement makes the redirection gateway server discussed in this section more important in a hybrid cloud environment.

4.2.2 Clustering of Individual reservations

The following section talks of ways of clustering individual reservations of VCL nodes.

4.2.2.1 Clustering using Torque resource manager and Maui scheduler

The report already talks about this in chapter 3. If we are able to get dedicated high end machines for HPC clusters in VCL, we can realize this implementation with minimum performance degradation.

4.2.2.2 Clustering using Rocks

Rocks, is a linux distribution intended for High Performance compute clusters. It is based on CentOS. It has a modified installer (a slightly different version of Anaconda) which is capable of bringing up the entire cluster of nodes. Rocks installation consists of installing a master node the first time. Then the master node can detect the compute nodes in the network and install the rocks client rolls on them via PXE.

As shown in the Figure 60, Rocks has a front-end server. This is the master server that we install at the start. The master server has two interfaces, one for interfacing with the external traffic or connecting to the users. The other interface connects to an internal switch connecting all the compute nodes. The master node has a bootup script which it runs while booting the compute nodes. The master node identifies the remaining nodes in the network and the administrator can now bring up the compute nodes easily.

The compute nodes as well as the master nodes comes with pre-installed MPI libraries, torque, MAUI scheduler, etc. Thus we have a high performance compute cluster up and running. Users can now compile their code on the front-end servers and queue their jobs to the scheduler. The scheduler selects a particular node (or a number of nodes requested by user) for the execution of the user job and returns the output back to the users. To enhance data sharing capabilities a NFS setup between the front-end nodes and the back-end compute nodes can be configured. This makes it easier for data maintenance and movement.

In this report, we explore the possibility if Rocks cluster can be made dynamic so as to integrate it with VCL theoretically. If the master server can control the compute node installation across a user select number of nodes in VCL, we can easily create flexible aggregated clusters.

High level work flow of cluster installation:

i. VCL will provision a master node onto a blade server.

ii. VCL will reserve 'n' more servers as specified by the user as compute nodes.
iii. VCL will provide the master node with the details of the compute nodes. It is necessary for the compute nodes to be in the same blade server or atleast connected to the same network switch (this can be controlled in VCL with special arrangements).
iv. VCL needs to allow the master node to PXE install new rolls/images on these servers.
v. Once the master node discovers the compute nodes, the installation script given to the modified Anaconda installer will install and bring up the compute nodes.
vi. The blade servers already have two Ethernet ports one used for external traffic and other for internal. The user can see the external interfaces for the nodes in a VCL environment but we can arrange for restrictions on user access via these interfaces.
vii. The master node and compute nodes can use one of these interfaces for cluster and storage traffic while the other interface can be used for MPI traffic.

Thus, theoretically we can have Rocks clusters working with VCL. Figure 60 shows the architecture of Rocks Cluster in VCL. Rocks can install and boot up a cluster with a certain number of nodes. The master node can be provisioned via VCL. The VCL can provide a kickstart file to the master node with the number of slave nodes suggested by the user to bring up the remaining cluster. This will enable a flexible cluster environment in VCL. This setup is hard in an on-demand environment. Since the Rocks cluster needs PXE install, it necessary for the nodes to be on the same chassis as the master node.

In an on-demand cloud environment, this is rather a restriction on the resource flexibility that a cloud solution promises. There can be special ways in which VCL can be notified to provide for 'n' machines on the same chassis, but there is a possibility that VCL has already provisioned other machines on the chassis and there are not enough blades to service the request. If a particular number of chassis and blade servers can be reserved for Rocks cluster then it is similar to having several in-house clusters. There are similar other clustering solutions which can be better than Rocks e.g. perceus [6]. We do not have a practical setup which works with Rocks and the exercise is left as future work. The report explores another

way of enabling HPC in cloud using the IBM BlueGene/P which is explained in the next section.

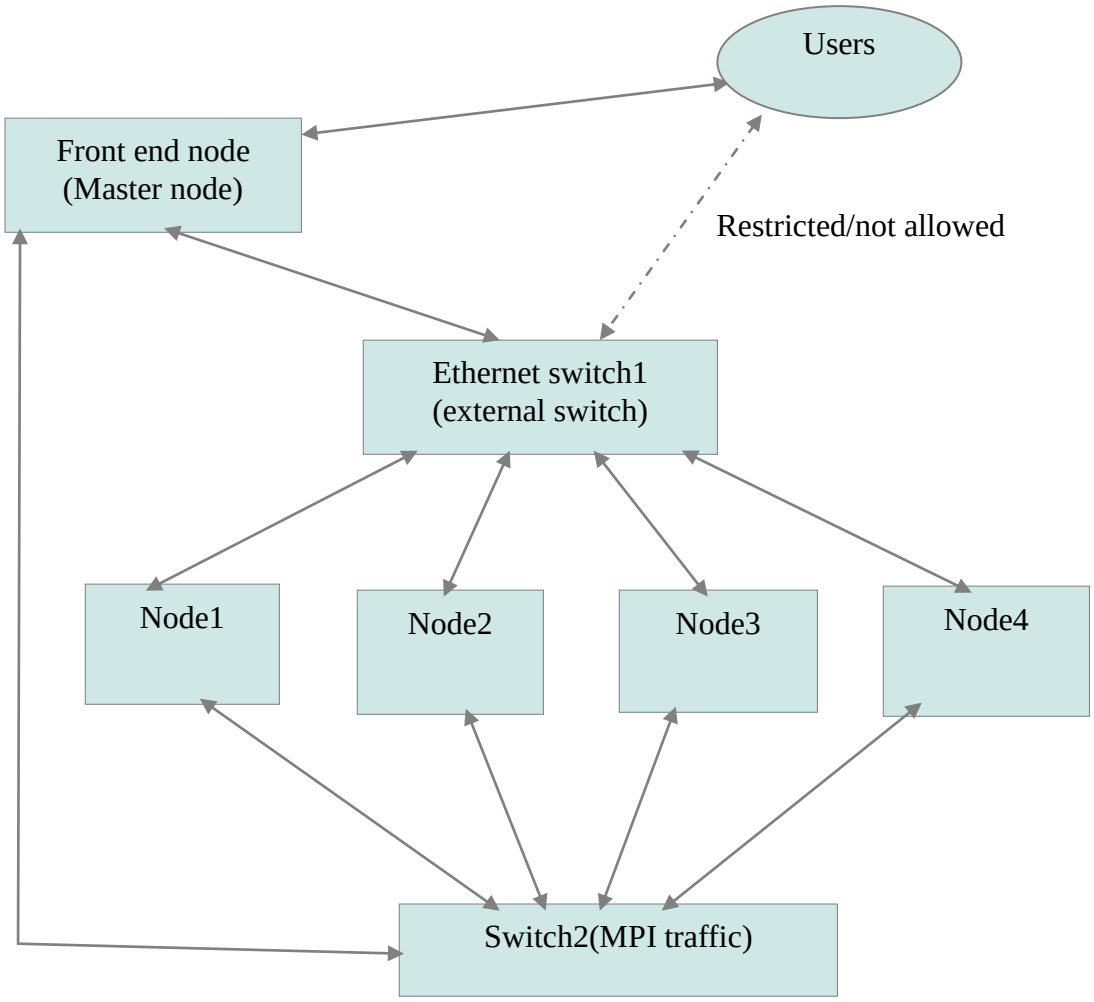

Figure 60: Rocks Cluster architecture in VCL

Chapter 5

Next Generation Hybrids

With the growth in data influx and analytics driven decision making, HPC and more recently HPD are becoming a common part of complex data and information management workflows.. In previous chapters we examined some HPC options available to VCL users today – VCL-HPC (a preconfigured HPC solution), VCL-GP clusters, re-direction based access to differentiated HPC facilities. The first option works very well. The second option works well provided that those who construct dynamic clusters have hardware-as-a-service privileges so that they can explicitly configure both computational node profiles as well as interconnection topologies. The third option works very well, but because access is indirect, and the resources are differentiated, the user has limited control over those resources and may not be able to fully serve their needs. An interesting solution would be one where one could have finer control over very high-performance resource elements – such as GPU cores, or IBM BlueGene/P nodes, in order to construct a pool of HPC resources that VCL users could compose into HPC clusters.

In the following section we explore the possibility and difficulties in implementing a multi-GPU environment (CPU not involved in computations) which can achieve higher stability than GPU+hostCPU environments. We also examine what it would take to integrate IBM BG/P nodes with VCL resources.

5.1 Multi-GPU in a network

The idea behind multi-GPU solution is to design a wrapper library around the existing CUDA library using SunRPC or MPI. SunRPC and MPI provide a framework for calling remote methods and also to send data to remote components of a cluster. This would enable users to have a homogeneous view of the GPGPU's. GPGPU could be viewed as a single compute resource rather than as a peripheral device attached to each node in the cluster.

Several of the GPU cards could be viewed as a pool of GPU processors rather than separate devices. The design and implementation for such a system using MPI has been discussed in [103]. The CPU on the hosting machine can be utilized for transferring the data along the network (to and from), and that CPU would be responsible for the exchange of cluster messages. The GPGPU's would be utilized solely for data intensive parallel computations. While host CPU takes the responsibility for data transfers and manages the MPI message exchanges. An MPI based implementation can also utilize the shared memory abstraction that MPI provides. The wrapper library described in [103] is a viable option and can open new research areas for multi-GPU environments. Since VCL-HPC utilizes an MPI bus and a private image loading and control bus, we could extend the VCL-HPC concept to encompass such a design.

A similar deep research was done by M. Strengert et al. in [60] to implement a multi-GPU model for GPU's (called CUDASA) on the same node as well as on the network. They have defined new compiler constructs and implemented a source level compiler which converts these new constructs into simple CUDA constructs. The implementation provides a multi-GPU environment with the programmer transparent to the underlying library calls which convert the higher level constructs into CUDA and MPI functions. This implementation spawns threads or MPI jobs on the node and over the network respectively. Each such thread/job polls the job queue for distributed jobs/threads generated by the CUDASA library. They do not guarantee deterministic assignment of GPU cards to CUDA kernels.

We can design for implementing a simple multi-GPU environment with deterministic mapping of CUDA kernels to GPU's in the network. What this means is that the programmer has to understand the network topology and map the CUDA kernels to specific GPUs in the network. For this scenario the GPUs in the network need some kind of addressing. The design for such a system has been discussed in [103].

Volodymyr V. Kindratenko et al. in [103] explain how GPU’s in a network can be virtualized using a global addressing scheme. This wrapper library can be used to build more generic

multi-GPU clusters. It will be interesting to see if these multi-GPU wrapper libraries can actually enhance the usability of multi-GPU configurations. Some intelligent tweaks to the HPL algorithm have been suggested in [104] for enhancing the performance of HPL on multi-GPU clusters. We have not tried or tested this algorithm as part of this thesis work. Understanding the performance of this new HPL algorithm on multi-GPU clusters (ARC cluster at NCSU) would be interesting but we leave this as a part of future research.

5.2 Hybrid cluster provisioning on IBM BlueGene/P

It would be convenient and beneficial, if we could integrate real supercomputers into a cloud – perhaps partition an IBM BlueGene/P (BG/P) into original BG engine, and a set of BG processors that VCL could individually and dynamically manipulate through one of VCL management nodes. This would allow the users to dynamically construct topologies and sub-clouds based on ultra high performance nodes and interconnects which map to the structure of the software instead of forcing the software to map to the hardware architecture.

The Kitty-Hawk project [61] enables such a possibility. IBM recently released open source drivers which allow unmodified Linux to run on the BG/P architecture. This release has interested a lot of computational scientists and a project is in progress at the Argonne National Labs [100] to use BG/P as a High Performance cluster which runs Linux. This opens up the possibility of integrating BG/P and VCL using Kitty-Hawk.

“A Blue Gene/P node is composed of four 850 MHz cache-coherent PowerPC cores in a system-on-a-chip with DRAM and interconnect controllers. The nodes are grouped in units of 32 on a node card via the interconnects and the node cards are grouped in units of 16 onto a midplane. There are 2 midplanes in a rack providing a total of 1024 nodes and, in its current configuration, a total of 2TB of RAM. Multiple racks can be joined together to construct an installation. The theoretical hardware limit to the number of racks is 16,384. The maximum number of nodes is 16.7 million with 67.1 million cores and 32 Petabytes of memory. Each node card can have up to two I/O nodes = the same basic unit but with an

additional 10G Ethernet port. Hence, each rack has an external I/O bandwidth of up to 640 Gbit/s; the aggregate I/O bandwidth of the maximum installation is 10.4 Petabits/s" [61].

There are four relevant networks on the Blue Gene/P:

i. "A global secure control network: This provides the network interconnect for the boot up process. This network helps in the IPMI communications (power on, power off, etc) as well as helps the TFTP communication needed for remote operating system installation.
ii. A hierarchical collective network: This superimposed network provides broadcast and multicast communication.
iii. A 3-D torus network: This it main point to point interconnect connecting all the nodes. Each node has six outgoing and six incoming connections, 3.4 Gbits/s each. Hence the network bandwidth scales up to a theoretical limit of 40.8 Gbits/s for each node.
iv. A 10 Gbit Ethernet for external communication."[61]

The Kitty-hawk project [61] enables running Linux machines on the BG/P architecture. BG/P architecture provides seamless clustering of Linux based compute nodes with fast interconnects with support for RDMA across these machines. A basic design to exploit the BG/P and Kitty-Hawk capabilities in VCL is discussed [62].

The idea is to build a controller node, essentially a VCL management node with a BG/P provisioning module, to launch KittyHawk and control the BG/P compute nodes and topologies. A BG/P compute node is a system on chip (SOC) and does not have any local storage capabilities. To boot an image on the BG/P compute node, the bootstrap root file system image, ramdisk, is either pushed to the compute node in the Push Model, or it is picked up by the compute node by booting up from a shared NAS/NFS in a Pull Model.

The controller node, thus, has the ability to control or provision compute nodes. To enable BG/P compute node provisioning in VCL, the controller node must be able to talk to other

VCL management nodes via some interface. What this means is that the controller node, and possibly individual BG/P nodes, is registered to VCL management node resident on BG/P.

When a BG/P image is requested for reservation, the management node will first talk to the controller node running on the BG/P and establish a secure connection and forward the request to the controller node.

A new provisioning engine needs to be built which understands the KittyHawk controller API's. These API's will request compute nodes from the controller node running on BG/P.
The controller node running on BG/P will transform these VCL API requests into equivalent Kitty-Hawk commands. These commands might inform Kitty-Hawk about the provisioning details such as the number of nodes to be used, the root file system image to be used across these nodes and other information such as the number of network interfaces each node would have, their interconnect topology, the secondary storage to be used to store output, etc. [62]. Figure 61 illustrates the solution.

We could also use a different approach. A VCL redirection server discussed in chapter 4 could gateway into the BG/P cluster, launch Kitty-Hawk and then talk directly to the BG/P node controller. The abstraction provided by this server would ensure a secure and isolated access to the BlueGene/P clusters which are located at the Argonne National Labs. As we explore more options for HPC in cloud, more use cases are generated for the abstract gateway server explained in chapter 4. This is still an open research topic.

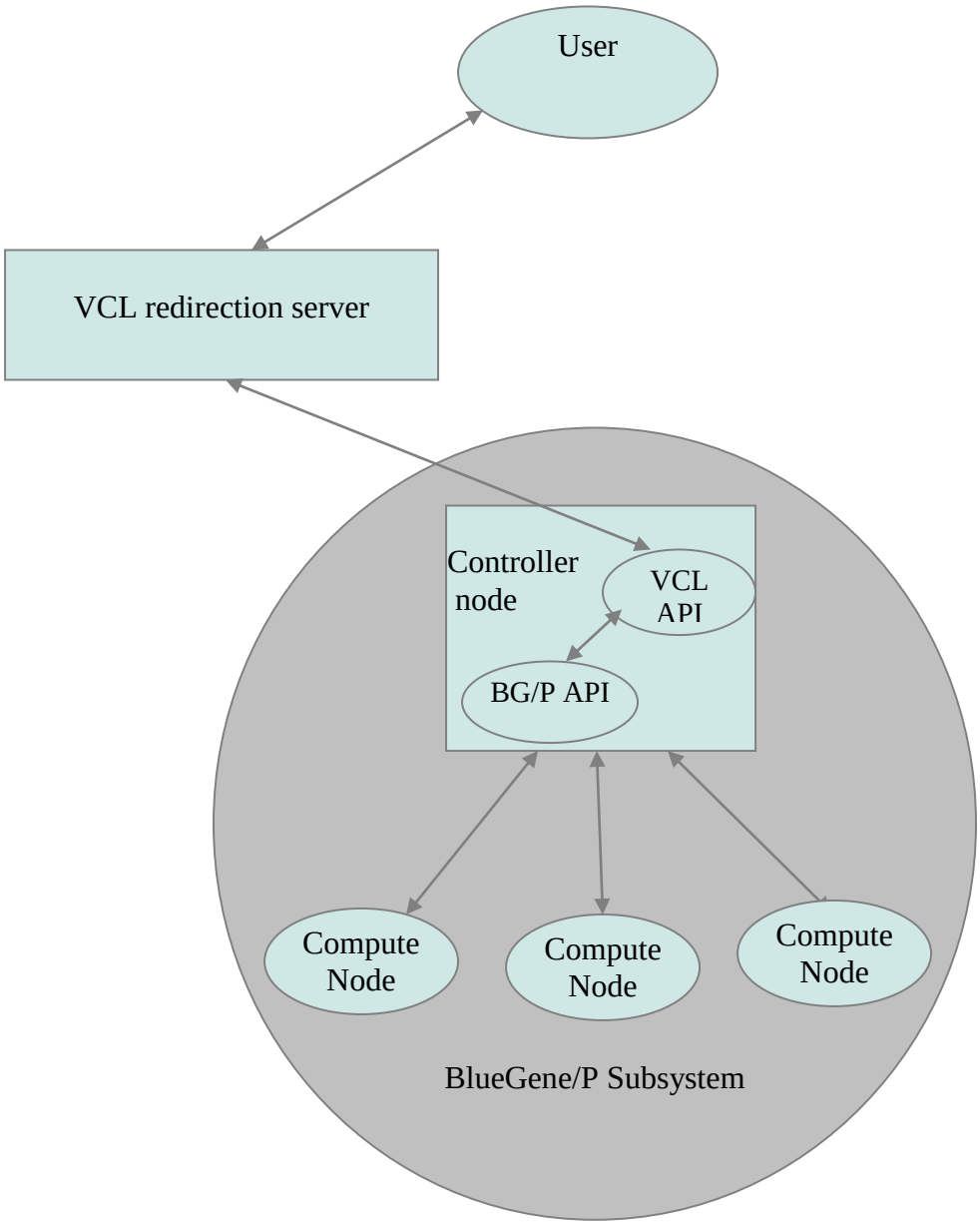

Figure 61: BlueGene/P provisioning module in VCL

Chapter 6
Conclusion

This thesis has taken a fresh look at the NCSU VCL HPC capabilities as they exist now, and at possible extension of these capabilities through hybrid solutions. There are four types of HPC resources available to NCSU researchers: 1) centrally managed VCL affiliated resources, 2) specialized HPC resources open to NCSU community (such as specialized GPGPU cluster), 3) private (research specific) HPC facilities and clusters, and 4) external resources (e.g., ORNL supercomputers). VCL affiliated resources are of two types – preconfigured VCL-HPC facilities, and user constructed clusters using VCL-GP resources. We also explore tighter integration of specialized HPC facilities, such as GPUs and BlueGene/P resources, into VCL – beyond the currently available option of access via re-directions.

As part of the study we examined performance of the VCL and specialized resources. For example, "in-house" InfiniBand interconnected HPC clusters perform very well. Our measurements indicate that a 32 node ARC cluster (512 AMD cores, IB interconnect) is able to achieve about 2.3 TFLOPS with HPL benchmark. Similarly, parts of VCL-HPC facilities appear to be able to achieve as much as 392 GFLOPS with HPL benchmark executed on 64 cores (IB interconnect). Nevertheless, there are issues. For example,

- Occasional interconnect congestion
- Node memory and disk space limitations
- Software library issues with multithreading and their interoperability
- Compatibility of compilers and certain libraries with respect to performance

Large variability is seen in the VCL-GP and the ARC CPU+GPU configurations. The performance variability as observed form the experiments may not be acceptable for next generation HPC and HPD applications. The performance variability in CPU+GPU in a clustered environment is low in terms of the total capacity offered by the GPU co-processor

unit. The reasons for this might be in the design of the distributed algorithm or the communication interconnects. One of the reasons also might be the inefficient use of multi-GPU configurations. We explored current research activities which could enhance the future of multi-GPU clusters.

A possible solution to the problem is dynamic access to specialized VCL clusters based on in-house as well as external resources. The dynamic access could be engineered in an intelligent way by providing additional abstraction based on access redirection methodologies. A good example in this context is the project “elastic Nimbus” [50]. In addition, this work looked at the possibility of integrating specialized clusters or cluster of supercomputer nodes like the IBM BlueGene/P with VCL.

This research finds that VCL and specialized clusters like BlueGene/P and multi-GPU can be combined into hybrid structures that jointly may be able to provide a broader range of high performing topologies suitable for different problems and complex workflows. For example, either simultaneous or parallel access to both tightly and loosely coupled HPC and HPD facilities. A good integrated solution may be similar to elastic Nimbus-like [50] architecture with the Nimbus slave resource manager corresponding to a VCL management node.

Traditional methods of constructing a HPC using VCL-GP do not work well. HPC in a cloud needs to be constructed by intelligently engineering hybrid structures, such as it is done in VCL-HPC and ARC cluster. We conclude that HPC in a cloud is viable and its future is promising provided the constructs are properly engineered.

REFERENCES

[1] "Amazon Elastic Compute Cloud (Amazon EC2)." *Amazon Web Services*. Web. 17 Mar. 2012. <http://aws.amazon.com/ec2/>.

[2] "HPC - NC State." *North Carolina State University*. Web. 17 Mar. 2012. <http://www.ncsu.edu/itd/hpc/main.php>.

[3] "ARC: ARoot Cluster for Research into Scalable Computer Systems." *ARC Cluster*. Web. 17 Mar. 2012. <http://moss.csc.ncsu.edu/~mueller/cluster/arc/>.

[4] "Cluster Workshop - Build Your Own Rocks Cluster." *Theoretical and Computational Biophysics Group*. Web. 17 Mar. 2012. <http://www.ks.uiuc.edu/Training/Workshop/Cluster/files/rocks.html>.

[5] "Www.rocksclusters.org | Rocks Website." Web. 17 Mar. 2012. <http://www.rocksclusters.org/wordpress/>.

[6] "Perceus Website." *Home*. Web. 17 Mar. 2012. <http://www.perceus.org/site/html/debian_quickstart.html>.

[7] "Delivering Results Faster, On Demand." *Services*. Web. 17 Mar. 2012. <http://www.nimbix.net/services>.

[8] "Fully Managed Hosting Made Easy." *Cloud Hosting, Managed Hosting and Colocation Services*. Web. 17 Mar. 2012. <http://www.peer1.com/>.

[9] "Cloud Services for GPU Computing." *GASS*. Web. 17 Mar. 2012. <http://www.hoopoe-cloud.com/Architecture.aspx>.

[10] Antoine Petitet, R. Clint Whaley, Jack J. Dongarra, and Andy Cleary. HPL { A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Com-puters. Innovative Computing Laboratory, September 2000. Available at <http://icl.cs.utk.edu/hpl/> and <http://www.netlib.org/benchmark/hpl/>

[11] "Software/dbus." *Freedesktop.org* -. Web. 17 Mar. 2012. <http://www.freedesktop.org/wiki/Software/dbus>.

[12] "D-Bus: Main Page." Web. 17 Mar. 2012. <http://dbus.freedesktop.org/doc/api/html/index.html>.

[13] Wang F, Yang CQ, Du YF et al. Optimizing linpack benchmark on GPU-accelerated petascale supercomputer. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(5): 854–865 Sept. 2011. DOI 10.1007/s11390-011-0184-1 Wang, Feng, Can-Qun Yang, Yun-Fei Du, Juan Chen, Hui-Zhan Yi, and Wei-Xia Xu. "Optimizing Linpack Benchmark on GPU-Accelerated Petascale Supercomputer." *Journal of Computer Science and Technology* 26.5 (2011): 854-65. Print.

[14] *NVIDIA*. 2009. Web. 17 Mar. 2012. <NVIDIA's Next Generation CUDA TM Compute Architecture, Fermi TM>.

[15] *CUDA C Programming Guide*. 2010. Web. 2011. <http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf>.

[16] Barrett, Daniel J., and Richard E. Silverman. *SSH, the Secure Shell: The Definitive Guide*. Cambridge [Mass.: O'Reilly, 2001. Print.

[17] Peng Di; Qing Wan; Xuemeng Zhang; Hui Wu; Jingling Xue; , "Toward Harnessing DOACROSS Parallelism for Multi-GPGPUs," Parallel Processing (ICPP), 2010 39th International Conference on , vol., no., pp.40-50, 13-16 Sept. 2010 doi: 10.1109/ICPP.2010.13

[18] Zhe Fan; Feng Qiu; Kaufman, A.; Yoakum-Stover, S.; , "GPU Cluster for High Performance Computing," Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference , vol., no., pp. 47, 06-12 Nov. 2004 doi: 10.1109/SC.2004.26

[19] Pechanec, Jan. "How the SCP Protocol Works." Weblog post. *Jan Pechanec's Weblog*. Oracle, 9 July 2007. Web. 2011. <https://blogs.oracle.com/janp/entry/how_the_scp_protocol_works>.

[20] "Rocks Cluster Distribution: Users Guide." Web. 2011. <http://www.rocksclusters.org/rocks-documentation/4.1/>.

[21] "NVIDIA Developer Zone." *NVIDIA Developer Zone*. Web. 2011. <http://developer.nvidia.com/>.

[22] Massimiliano Fatica. 2009. Accelerating linpack with CUDA on heterogenous clusters. In Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2). ACM, New York, NY, USA, 46-51. DOI=10.1145/1513895.1513901

[23] Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. 2008. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming (PPoPP '08). ACM, New York, NY, USA, 73-82. DOI=10.1145/1345206.1345220

[24] Dongarra J J, Luszczek P, Petitet A. The linpack benchmark: Past, present and future. Concurrency and Computation: Practice and Experience, 2003, 15(9): 803-820.

[25] "Research/Implementation Project Proposal: Automatic Tuning of the High Performance Linpack Benchmark." *Automatic Tuning of the High Performance Linpack Benchmark*. Web. 17 Mar. 2012. <http://cs.anu.edu.au/~Peter.Strazdins/postgrad/AutoTunedHPL.html>.

[26] Jack J. Dongarra, J. Bunch, Cleve Moler, and G. W. Stewart. LINPACK User's Guide. SIAM, Philadelphia, PA, 1979.

[27] Yan Zhai; Mingliang Liu; Jidong Zhai; Xiaosong Ma; Wenguang Chen; , "Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications," High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for , vol., no., pp.1-10, 12-18 Nov. 2011

[28] Carlyle, A.G.; Harrell, S.L.; Smith, P.M.; , "Cost-Effective HPC: The Community or the Cloud?," Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on , vol., no., pp.169-176, Nov. 30 2010-Dec. 3 2010 doi: 10.1109/CloudCom.2010.115

[29] Deelman, E.; Singh, G.; Livny, M.; Berriman, B.; Good, J.; , "The cost of doing science on the cloud: The Montage example," High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for , vol., no., pp.1-12, 15-21 Nov. 2008 doi: 10.1109/SC.2008.5217932

[30] C. Evangelinos and C. Hill. Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In The 1st Workshop on Cloud Computing and its Applications (CCA), 2008.

[31] Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy, and Tom McGlynn. 2010. Case study for running HPC applications in public clouds. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10). ACM, New York, NY, USA, 395-401. DOI=10.1145/1851476.1851535

[32] Hemsoth, Nicole. "HPC in the Cloud: Amazon Adds HPC Capability to EC2." *HPC in the Cloud: Dedicated to Covering High-end Cloud Computing in Science, Industry and the Datacenter*. Web. 17 Mar. 2012. <http://www.hpcinthecloud.com/hpccloud/2010-07-13/amazon_adds_hpc_capability_to_ec2.html>.

[33] Hoffa, C.; Mehta, G.; Freeman, T.; Deelman, E.; Keahey, K.; Berriman, B.; Good, J.; , "On the Use of Cloud Computing for Scientific Workflows," eScience, 2008. eScience '08. IEEE Fourth International Conference on , vol., no., pp.640-645, 7-12 Dec. 2008 doi: 10.1109/eScience.2008.167

[34] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K. Panda. 2006. A case for high performance computing with virtual machines. In Proceedings of the 20th annual international conference on Supercomputing (ICS '06). ACM, New York, NY, USA, 125-134. DOI=10.1145/1183401.1183421

[35] Iosup, A.; Ostermann, S.; Yigitbasi, M.N.; Prodan, R.; Fahringer, T.; Epema, D.H.J.; , "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing," Parallel and Distributed Systems, IEEE Transactions on , vol.22, no.6, pp.931-945, June 2011 doi: 10.1109/TPDS.2011.66

[36] . Jackson, K.R.; Ramakrishnan, L.; Muriki, K.; Canon, S.; Cholia, S.; Shalf, J.; Wasserman, H.J.; Wright, N.J.; , "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on* , vol., no., pp.159-168, Nov. 30 2010-Dec. 3 2010

[37] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P. Berman, and Phil Maechling. 2010. Data Sharing Options for Scientific Workflows on Amazon EC2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (SC '10). IEEE Computer Society, Washington, DC, USA, 1-9

[38] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In The 1st Workshop on Cloud Computing and its Applications (CCA), 2008.

[39] Jie Li; Humphrey, M.; Agarwal, D.; Jackson, K.; van Ingen, C.; Youngryel Ryu; , "eScience in the cloud: A MODIS satellite data reprojection and reduction pipeline in the Windows Azure platform," *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on* , vol., no., pp.1-10, 19-23 April 2010

[40] Heshan Lin; Balaji, P.; Poole, R.; Sosa, C.; Xiaosong Ma; Wu-chun Feng; , "Massively parallel genomic sequence search on the Blue Gene/P architecture," *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for* , vol., no., pp.1-11, 15-21 Nov. 2008

[41] Jeffrey Napper and Paolo Bientinesi. 2009. Can cloud computing reach the top500?. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop* (UCHPC-MAW '09). ACM, New York, NY, USA, 17-20.

[42] "High Performance Computing (HPC) on AWS." *High Performance Computing*. Web. 17 Mar. 2012. <http://aws.amazon.com/hpc-applications/>.

[43] Mayur R. Palankar, Adriana Iamnitchi, Matei Ripeanu, and Simson Garfinkel. 2008. Amazon S3 for science grids: a viable solution?. In *Proceedings of the 2008 international workshop on Data-aware distributed computing* (DADC '08). ACM, New York, NY, USA, 55-64.

[44] "Home | TOP500 Supercomputing Sites." *Home*. Web. 17 Mar. 2012. <http://www.top500.org/>.

[45] T. University. Technique report r2011.4.10. <http://www.hpctest.org.cn/resources/cloud.pdf.>

[46] Vecchiola, C.; Pandey, S.; Buyya, R.; , "High-Performance Cloud Computing: A View of Scientific Applications," *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on* , vol., no., pp.4-16, 14-16 Dec. 2009

[47] Jackson, K.R.; Ramakrishnan, L.; Muriki, K.; Canon, S.; Cholia, S.; Shalf, J.; Wasserman, H.J.; Wright, N.J.; , "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on* , vol., no., pp.159-168, Nov. 30 2010-Dec. 3 2010 doi: 10.1109/CloudCom.2010.69

[48] Hill, Z.; Humphrey, M.; , "A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?," *Grid Computing, 2009 10th IEEE/ACM International Conference on* , vol., no., pp.26-33, 13-15 Oct. 2009

[49] Sterling, T.; Stark, D.; , "A High-Performance Computing Forecast: Partly Cloudy," *Computing in Science & Engineering* , vol.11, no.4, pp.42-49, July-Aug. 2009

[50] Marshall, Paul; Keahey, Kate; Freeman, Tim; , "Elastic Site: Using Clouds to Elastically Extend Site Resources," *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* , vol., no., pp.43-52, 17-20 May 2010

[51] Mingliang Liu, Jidong Zhai, Yan Zhai, Xiaosong Ma, and Wenguang Chen. 2011. One optimized I/O configuration per HPC application: leveraging the configurability of cloud. In *Proceedings of the Second Asia-Pacific Workshop on Systems* (APSys '11). ACM, New York, NY, USA, , Article 15 , 5 pages

[52] Open Cloud Infrastructure. Simply Scaled. Production Ready." *Our Solution*. Web. 17 Mar. 2012. <http://www.cloudscaling.com/solution/>

[53] Private Cloud for High Performance Computing (HPC) Applications." *Nimbula*. Web. 17 Mar. 2012. <http://nimbula.com/solutions/enterprise/hpc-applications/>

[54] "GPU Cloud Computing Service Providers." *Document Moved*. Web. 17 Mar. 2012. <http://www.nvidia.com/object/gpu-cloud-computing-service.html>.

[55] "Frequent Asked Questions on the LINPACK Benchmark." *The Netlib*. Web. 17 Mar. 2012. <http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html>.

[56] “Performance of Various Computers Using Standard Linear Equations Software”, Jack Dongarra, University of Tennessee, Knoxville TN, 37996, Computer Science Technical Report Number CS - 89 – 85, today’s date, url:http://www.netlib.org/benchmark/performance.ps.

[57] "The Linpack Benchmark." *Home*. Web. 17 Mar. 2012. <http://www.top500.org/project/linpack>.

[58] "QUDA: A Library for QCD on GPUs." *QUDA*. Web. 17 Mar. 2012. <http://lattice.github.com/quda/>.

[59] M.A. Clark, R. Babich, K. Barros, R.C. Brower, C. Rebbi, Solving lattice QCD systems of equations using mixed precision solvers on GPUs, Computer Physics Communications, Volume 181, Issue 9, September 2010, Pages 1517-1528, ISSN 0010-4655, 10.1016/j.cpc.2010.05.002.

[60] M. Strengert, C. Müller, C. Dachsbacher, and T. Ertl. "CUDASA: Compute Unified Device and Systems Architecture." Ed. J. Favre, K. - L. Ma, and D.

Weiskopf.*Eurographics Symposium on Parallel Graphics and Visualization* (2008). Print. <http://www.vis.uni-stuttgart.de/dachsbacher/download/egpgv08.pdf>

[61] Jonathan Appavoo, Volkmar Uhlig, and Amos Waterland. 2008. Project Kittyhawk: building a global-scale computer: Blue Gene/P as a generic computing platform. *SIGOPS Oper. Syst. Rev.*42, 1 (January 2008), 77-84.

[62] Salil Kanitkar, Hybrid Cloud Computing, VCL and BlueGene/P. Internal report, at North Carolina State University: under the guidance of Dr. Mladen Vouk, Fall 2011

[63] Preeti Mupalla, Ashwini Narasimhamurthy, Kaushal Didwania, Project Report: High Performance Computing in a Cloud, Internal report, at North Carolina State University: under the guidance of Dr. Mladen Vouk, CSC-591-006, Spring 2011.

[64] Stuart, J.A.; Owens, J.D.; , "Message passing on data-parallel architectures," *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on* , vol., no., pp.1-12, 23-29 May 2009

[65] Nieplocha, J.; Harrison, R.J.; Littlefield, R.J.; , "Global Arrays: a portable “shared-memory” programming model for distributed memory computers," *Supercomputing '94. Proceedings* , vol., no., pp.340-349, 816, 14-18 Nov 1994

[66] Fan, Zhe, Qiu, Feng, Kaufman, E. Arie,” Zippy: A Framework for Computation and Visualization on a GPU Cluster “ Computer Graphics Forum, Vol. 27, No. 2. (April 2008)

[67] "Home | Virtual Computing Lab (VCL)." *Home*. Web. 17 Mar. 2012. <http://vcl.ncsu.edu/>.

[68] "TORQUE Admin Manual." *TORQUE Administrator's Manual*. Web. 17 Mar. 2012. <http://www.clusterresources.com/torquedocs/index.shtml>.

[69] "Maui Cluster Schedulerâ„¢." *Cluster Resources*. Web. 17 Mar. 2012. <http://www.clusterresources.com/products/maui-cluster-scheduler.php>.

[70] "Open MPI: Open Source High Performance Computing." *Open MPI: Open Source High Performance Computing*. Web. 17 Mar. 2012. <http://www.open-mpi.org/>.

[71] "Welcome to Apacheâ„¢ Hadoopâ„¢!" *Welcome to Apacheâ„¢ Hadoopâ„¢!* Web. 17 Mar. 2012. <http://hadoop.apache.org/>.

[72] "Beowulf Cluster Computing with Linux, Second Edition." *Linux Journal*. Web. 17 Mar. 2012. <http://www.linuxjournal.com/article/7494>.

[73] Zhang, Yongpeng, and Frank Muller. "Auto-Generation and Auto-Tuning of 3D Stencil Codes on GPU Clusters." in International Symposium on Code Generation and Optimization, Apr 2012 <http://moss.csc.ncsu.edu/~mueller/publications.html>.

[74] Qihang Huang; Zhiyi Huang; Werstein, P.; Purvis, M.; , "GPU as a General Purpose Computing Resource," Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008. Ninth International Conference on , vol., no., pp.151-158, 1-4 Dec. 2008 doi: 10.1109/PDCAT.2008.38

[75] "About :: GPGPU.org." *General-Purpose Computation on Graphics Processing Units*. Web. 17 Mar. 2012. <http://gpgpu.org/about>.

[76] Makris, P.; Skoutas, D.N.; Rizomiliotis, P.; Skianis, C.; , "A User-Oriented, Customizable Infrastructure Sharing Approach for Hybrid Cloud Computing Environments," Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on , vol., no., pp.432-439, Nov. 29 2011-Dec. 1 2011 doi: 10.1109/CloudCom.2011.64

[77] "HPC in the Cloud: Dedicated to Covering High-end Cloud Computing in Science, Industry and the Datacenter." *HPC in the Cloud: Dedicated to Covering High-end Cloud Computing in Science, Industry and the Datacenter*. Web. 17 Mar. 2012. <http://www.hpcinthecloud.com/>.

[78] Chari, Srini. "HPC in the Cloud: IBM Outlines Benefits of Engineering Clouds." *HPC in the Cloud: Dedicated to Covering High-end Cloud Computing in Science, Industry and the Datacenter*. June 2011. Web. 17 Mar. 2012.

<http://www.hpcinthecloud.com/hpccloud/2011-07-28/ibm_outlines_benefits_of_engineering_clouds.html>.

[79] "HPC Cloud: Supercomputing to Go." - *E & T Magazine*. Web. 17 Mar. 2012. <http://eandt.theiet.org/magazine/2012/02/supercomputing-to-go.cfm>.

[80] "Eucalyptus Cloud Computing Software." *Cloud Computing Software from Eucalyptus*. Web. 17 Mar. 2012. <http://www.eucalyptus.com/>.

[81] "Open Stack." *OpenStack Compute Â» OpenStack Open Source Cloud Computing Software*. Web. 17 Mar. 2012. <http://openstack.org/projects/compute/>.

[82] Mladen Vouk, Andy Rindos, Sam Averitt, John Bass, Michael Bugaev, Aaron Peeler, Henry Schaffer, Eric Sills, Sarah Stein, Josh Thompson, Matthew P. Valenzisi, "Using VCL Technology to Implement Distributed Reconfigurable Data Centers and Computational Services for Educational Institutions," IBM Journal of Research and Development, Vol. 53, No. 4, pp. 2:1-18, 2009, b) Henry E. Schaffer, Samuel F. Averitt, Marc I. Hoit, Aaron Peeler, Eric D. Sills, and Mladen A. Vouk, NCSUs Virtual Computing Lab: A Cloud Computing Solution," IEEE Computer, pp. 94-97, July 2009.

[83] "Nimbus." *Nimbus*. Web. 17 Mar. 2012. <http://www.nimbusproject.org/>.

[84] "Beowulf HOWTO." *Ibiblio*. Web. 17 Mar. 2012. <http://www.ibiblio.org/pub/linux/docs/HOWTO/archive/Beowulf-HOWTO.html>.

[85] The LINPACK 1000x1000 benchmark program. (See <http://www.netlib.org/benchmark/1000d> for source code.)

[86] "Apache-VCL." *Home*. Web. 17 Mar. 2012. <http://vcl.ncsu.edu/apache-vcl>.

[87] "Cloud Hosting." *Rackspace Hosting*. Web. 17 Mar. 2012. <http://www.rackspace.com/>.

[88] "NASA - National Aeronautics and Space Administration." *NASA*. Web. 17 Mar. 2012. <http://www.nasa.gov>.

[89] "VMware Virtualization Software for Desktops, Servers & Virtual Machines for Public and Private Cloud Solutions." *VMware Virtualization Software for Desktops, Servers & Virtual Machines for Public and Private Cloud Solutions*. Web. 17 Mar. 2012. <http://www.vmware.com/>.

[90] "Navigation." *Texas Advanced Computing Center*. Web. 17 Mar. 2012. <http://www.tacc.utexas.edu/tacc-projects/gotoblas2>.

[91] Mladen A. Vouk, Eric Sills and Patrick Dreher, "Integration of High-Performance Computing into Cloud Computing Services," Ch 11 in Handbook of Cloud Computing, editors B.Furht and A.Escalante, Springer, ISBN: 978-1-4419-6523, pp.255-276, 2010

[92] "Project Kittyhawk: A Global Scale ComputerTowards a Public Utility Cloud Super Computer Platform." Web. 2012. <http://kittyhawk.bu.edu/kittyhawk/Kittyhawk.html>.

[93] Alam, S.; Barrett, R.; Bast, M.; Fahey, M.R.; Kuehn, J.; McCurdy, C.; Rogers, J.; Roth, P.; Sankaran, R.; Vetter, J.S.; Worley, P.; Yu, W.; , "Early evaluation of IBM BlueGene/P," *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for* , vol., no., pp.1-12, 15-21 Nov. 2008 doi: 10.1109/SC.2008.5214725

[94] Asim Nisar, Mongkol Ekpanyapong, Antonio C Valles, Kuppuswamy Sivakumar ‘‘Original 45nm Intel Core2 Processor Performance’’ Intel Technology Journal. http://www.intel.com/technology/itj/2008/v12i3/1-paper/1-abstract.htm (October 2008).

[95] "Cray Inc., The Supercomputer Company." Web. 2012. <http://www.cray.com/Home.aspx>.

[96] "Xen Hypervisor Project." *Xen Hypervisor*. Web. 2012. <http://xen.org/products/xenhyp.html>.

[97] "Math Kernel Library from Intel." *IntelÂ® Software Network*. Web. 2012. <http://software.intel.com/en-us/articles/intel-mkl/>.

[98] "The Leader in Cluster, Grid and Cloud Management Software, Platform LSF" *High Performance Computing*. Web. 2012. <http://www.platform.com/workload-management/high-performance-computing>.

[99] "HPC - Hardware." *North Carolina State University*. Web. 2012. <http://www.ncsu.edu/itd/hpc/Hardware/henry2_architecture.php>.

[100] "Argonne National Laboratory ... for a Brighter Future." *Argonne National Laboratory ... for a Brighter Future*. Web. 2012. <http://www.anl.gov/index.html>.

[101] "Ganglia Monitoring System." *Ganglia Monitoring System*. Web. 2012. <http://ganglia.sourceforge.net/>.

[102] Friedl, Steve. "Steve Friedl's Unixwiz.net Tech Tips." *An Illustrated Guide to SSH Agent Forwarding*. Web. 2012. <http://www.unixwiz.net/techtips/ssh-agent-forwarding.html>.

[103] Kindratenko, V.V.; Enos, J.J.; Guochun Shi; Showerman, M.T.; Arnold, G.W.; Stone, J.E.; Phillips, J.C.; Wen-mei Hwu; , "GPU clusters for high-performance computing," *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on* , vol., no., pp.1-8, Aug. 31 2009-Sept. 4 2009 doi: 10.1109/CLUSTR.2009.5289128

[104] Rohr, D.; Bach, M.; Kretz, M.; Lindenstruth, V.; , "Multi-GPU DGEMM and High Performance Linpack on Highly Energy-Efficient Clusters," *Micro, IEEE* , vol.31, no.5, pp.18-27, Sept.-Oct. 2011
doi: 10.1109/MM.2011.66

[105] "HPCC." *Innovative Computing Laboratory*. Web. 2012. <http://icl.cs.utk.edu/hpcc/>.

APPENDIX

Appendix A

SCP traces

A.1 Trace for ssh auth-agent forward (passed)

niks@linux-xqbf:~> ssh -vvv nvtalpal@152.1.13.157 ls -l | tee ssh-authagent
OpenSSH_5.8p2, OpenSSL 1.0.0e 6 Sep 2011
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug2: ssh_connect: needpriv 0
debug1: Connecting to 152.1.13.157 [152.1.13.157] port 22.
....
....
....
debug1: Offering RSA public key: /home/niks/.ssh/id_rsa
debug3: send_pubkey_test
debug2: we sent a publickey packet, wait for reply
debug1: Remote: Forced command: bash -c 'ssh -l nvtalpal arc.csc.ncsu.edu
${SSH_ORIGINAL_COMMAND:-}'
debug1: Server accepts key: pkalg ssh-rsa blen 277
debug2: input_userauth_pk_ok: fp 95:90:72:47:85:cb:42:92:e1:c7:84:70:51:a3:0d:12
debug3: sign_and_send_pubkey: RSA 95:90:72:47:85:cb:42:92:e1:c7:84:70:51:a3:0d:12
debug1: Remote: Forced command: bash -c 'ssh -l nvtalpal arc.csc.ncsu.edu
${SSH_ORIGINAL_COMMAND:-}'
debug1: Authentication succeeded (publickey).
Authenticated to 152.1.13.157 ([152.1.13.157]:22).
debug2: fd 5 setting O_NONBLOCK
debug1: channel 0: new [client-session]
debug3: ssh_session2_open: channel_new: 0

debug2: channel 0: send open

debug1: Entering interactive session.

debug2: callback start

debug1: Requesting authentication agent forwarding.

debug2: channel 0: request auth-agent-req@openssh.com confirm 0

debug2: client_session2_setup: id 0

debug2: fd 3 setting TCP_NODELAY

debug3: packet_set_tos: set IP_TOS 0x08

debug1: Sending environment.

....

....

....

debug1: Sending command: ls -l

debug2: channel 0: request exec confirm 1

debug2: callback done

debug2: channel 0: open confirm rwindow 0 rmax 32768

debug2: channel 0: rcvd adjust 2097152

debug2: channel_input_status_confirm: type 99 id 0

debug2: exec request accepted on channel 0

debug1: client_input_channel_open: ctype auth-agent@openssh.com rchan 2 win 65536 max 16384

debug2: fd 7 setting O_NONBLOCK

debug3: fd 7 is O_NONBLOCK

debug1: channel 1: new [authentication agent connection]

debug1: confirm auth-agent@openssh.com

debug2: channel 1: rcvd eof

debug2: channel 1: output open -> drain

debug2: channel 1: obuf empty

debug2: channel 1: close_write

debug2: channel 1: output drain -> closed

debug1: channel 1: FORCE input drain

debug2: channel 1: ibuf empty

debug2: channel 1: send eof

debug2: channel 1: input drain -> closed

debug2: channel 1: send close

debug3: channel 1: will not send data after close

debug2: channel 1: rcvd close

debug3: channel 1: will not send data after close

debug2: channel 1: is dead

debug2: channel 1: garbage collecting

debug1: channel 1: free: authentication agent connection, nchannels 2

debug3: channel 1: status: The following connections are open:

#0 client-session (t4 r0 i0/0 o0/0 fd 4/5 cc -1)

#1 authentication agent connection (t4 r2 i3/0 o3/0 fd 7/7 cc -1)

total 364236

drwxr-xr-x 11 nvtalpal ncsu 4096 Mar 1 12:33 2hpl

....

....

....

debug1: fd 1 clearing O_NONBLOCK

Transferred: sent 4112, received 5672 bytes, in 1.1 seconds

Bytes per second: sent 3879.2, received 5350.9

debug1: Exit status 0

niks@linux-xqbf:~>

A.2 Trace for scp auth-agent forward (failed)

niks@linux-xqbf:~> scp -vvv a.c nvtalpal@152.1.13.157:/home/nvtalpal/.

Executing: program /usr/bin/ssh host 152.1.13.157, user nvtalpal, command scp -v -t --
/home/nvtalpal/.

OpenSSH_5.8p2, OpenSSL 1.0.0e 6 Sep 2011

debug1: Reading configuration data /etc/ssh/ssh_config

debug1: Applying options for *

debug2: ssh_connect: needpriv 0

debug1: Connecting to 152.1.13.157 [152.1.13.157] port 22.

....

....

....

debug1: Offering RSA public key: /home/niks/.ssh/id_rsa

debug3: send_pubkey_test

debug2: we sent a publickey packet, wait for reply

debug1: Remote: Forced command: bash -c 'ssh -l nvtalpal arc.csc.ncsu.edu

${SSH_ORIGINAL_COMMAND:-}'

debug1: Server accepts key: pkalg ssh-rsa blen 277

debug2: input_userauth_pk_ok: fp 95:90:72:47:85:cb:42:92:e1:c7:84:70:51:a3:0d:12

debug3: sign_and_send_pubkey: RSA 95:90:72:47:85:cb:42:92:e1:c7:84:70:51:a3:0d:12

debug1: Remote: Forced command: bash -c 'ssh -l nvtalpal arc.csc.ncsu.edu

${SSH_ORIGINAL_COMMAND:-}'

debug1: Authentication succeeded (publickey).

Authenticated to 152.1.13.157 ([152.1.13.157]:22).

debug2: fd 4 setting O_NONBLOCK

debug2: fd 5 setting O_NONBLOCK

debug1: channel 0: new [client-session]

debug3: ssh_session2_open: channel_new: 0

debug2: channel 0: send open

debug1: Entering interactive session.

debug2: callback start

debug2: client_session2_setup: id 0

debug2: fd 3 setting TCP_NODELAY

debug3: packet_set_tos: set IP_TOS 0x08

debug1: Sending environment.

....

....

....

debug1: Sending command: scp -v -t -- /home/nvtalpal/.

debug2: channel 0: request exec confirm 1

debug2: callback done

debug2: channel 0: open confirm rwindow 0 rmax 32768

debug2: channel 0: rcvd adjust 2097152

debug2: channel_input_status_confirm: type 99 id 0

debug2: exec request accepted on channel 0

debug2: channel 0: rcvd ext data 38

Permission denied, please try again.

debug2: channel 0: written 38 to efd 6

debug2: channel 0: rcvd ext data 38

debug2: channel 0: rcvd ext data 41

debug2: channel 0: rcvd eof

debug2: channel 0: output open -> drain

debug1: client_input_channel_req: channel 0 rtype exit-status reply 0

debug2: channel 0: rcvd close

debug2: channel 0: close_read

debug2: channel 0: input open -> closed

debug3: channel 0: will not send data after close

debug2: channel 0: obuf_empty delayed efd 6/(79)

Permission denied, please try again.

Permission denied (publickey,password).

debug2: channel 0: written 79 to efd 6

debug3: channel 0: will not send data after close

debug2: channel 0: obuf empty

debug2: channel 0: close_write

debug2: channel 0: output drain -> closed
debug2: channel 0: almost dead
debug2: channel 0: gc: notify user
debug2: channel 0: gc: user detached
debug2: channel 0: send close
debug2: channel 0: is dead
debug2: channel 0: garbage collecting
debug1: channel 0: free: client-session, nchannels 1
debug3: channel 0: status: The following connections are open:
  #0 client-session (t4 r0 i3/0 o3/0 fd -1/-1 cc -1)
debug1: fd 0 clearing O_NONBLOCK
debug1: fd 1 clearing O_NONBLOCK
Transferred: sent 3120, received 2632 bytes, in 0.2 seconds
Bytes per second: sent 13288.3, received 11209.9
debug1: Exit status 255
lost connection
niks@linux-xqbf:~>

A.3 Trace for scp auth-agent forward (passed)

niks@linux-xqbf:~> /opt/bin/scp -vvv a.c nvtalpal@152.1.13.157:/home/nvtalpal/.
Executing: program /opt/bin/ssh host 152.1.13.157, user nvtalpal, command scp -v -t /home/nvtalpal/.
OpenSSH_6.0-betap1, OpenSSL 1.0.0e 6 Sep 2011
debug1: Reading configuration data /opt/etc/ssh_config
debug2: ssh_connect: needpriv 0
debug1: Connecting to 152.1.13.157 [152.1.13.157] port 22.
....
....
....

debug1: Offering RSA public key: /home/niks/.ssh/id_rsa

debug3: send_pubkey_test

debug2: we sent a publickey packet, wait for reply

debug1: Remote: Forced command: bash -c 'ssh -l nvtalpal arc.csc.ncsu.edu

${SSH_ORIGINAL_COMMAND:-}'

debug1: Server accepts key: pkalg ssh-rsa blen 277

debug2: input_userauth_pk_ok: fp 95:90:72:47:85:cb:42:92:e1:c7:84:70:51:a3:0d:12

debug3: sign_and_send_pubkey: RSA 95:90:72:47:85:cb:42:92:e1:c7:84:70:51:a3:0d:12

debug1: Remote: Forced command: bash -c 'ssh -l nvtalpal arc.csc.ncsu.edu

${SSH_ORIGINAL_COMMAND:-}'

debug1: Authentication succeeded (publickey).

Authenticated to 152.1.13.157 ([152.1.13.157]:22).

debug2: fd 4 setting O_NONBLOCK

debug2: fd 5 setting O_NONBLOCK

debug1: channel 0: new [client-session]

debug3: ssh_session2_open: channel_new: 0

debug2: channel 0: send open

debug1: Entering interactive session.

debug2: callback start

debug1: Requesting authentication agent forwarding.

debug2: channel 0: request auth-agent-req@openssh.com confirm 0

debug2: client_session2_setup: id 0

debug2: fd 3 setting TCP_NODELAY

debug1: Sending command: scp -v -t /home/nvtalpal/.

debug2: channel 0: request exec confirm 1

debug2: callback done

debug2: channel 0: open confirm rwindow 0 rmax 32768

debug2: channel 0: rcvd adjust 2097152

debug2: channel_input_status_confirm: type 99 id 0

debug2: exec request accepted on channel 0

debug1: client_input_channel_open: ctype auth-agent@openssh.com rchan 2 win 65536 max 16384

debug2: fd 7 setting O_NONBLOCK

debug3: fd 7 is O_NONBLOCK

debug1: channel 1: new [authentication agent connection]

debug1: confirm auth-agent@openssh.com

debug2: channel 1: rcvd eof

debug2: channel 1: output open -> drain

debug2: channel 1: obuf empty

debug2: channel 1: close_write

debug2: channel 1: output drain -> closed

debug1: channel 1: FORCE input drain

debug2: channel 1: ibuf empty

debug2: channel 1: send eof

debug2: channel 1: input drain -> closed

debug2: channel 1: send close

debug3: channel 1: will not send data after close

debug2: channel 1: rcvd close

debug3: channel 1: will not send data after close

debug2: channel 1: is dead

debug2: channel 1: garbage collecting

debug1: channel 1: free: authentication agent connection, nchannels 2

debug3: channel 1: status: The following connections are open:

#0 client-session (t4 r0 i0/0 o0/0 fd 4/5 cc -1)

#1 authentication agent connection (t4 r2 i3/0 o3/0 fd 7/7 cc -1)

Sending file modes: C0644 55 a.c

debug2: channel 0: rcvd ext data 19

Sink: C0644 55 a.c

debug2: channel 0: written 19 to efd 6

```
a.c
100%   55     0.1KB/s   00:00
debug2: channel 0: read<=0 rfd 4 len 0
debug2: channel 0: read failed
debug2: channel 0: close_read
debug2: channel 0: input open -> drain
debug2: channel 0: ibuf empty
debug2: channel 0: send eof
debug2: channel 0: input drain -> closed
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug2: channel 0: rcvd eof
debug2: channel 0: output open -> drain
debug2: channel 0: obuf empty
debug2: channel 0: close_write
debug2: channel 0: output drain -> closed
debug2: channel 0: rcvd close
debug3: channel 0: will not send data after close
debug2: channel 0: almost dead
debug2: channel 0: gc: notify user
debug2: channel 0: gc: user detached
debug2: channel 0: send close
debug2: channel 0: is dead
debug2: channel 0: garbage collecting
debug1: channel 0: free: client-session, nchannels 1
debug3: channel 0: status: The following connections are open:
  #0 client-session (t4 r0 i3/0 o3/0 fd -1/-1 cc -1)
debug1: fd 0 clearing O_NONBLOCK
debug1: fd 1 clearing O_NONBLOCK
Transferred: sent 4384, received 3720 bytes, in 1.2 seconds
Bytes per second: sent 3616.3, received 3068.6
```

```
debug1: Exit status 0
niks@linux-xqbf:~>
```

Appendix B

Some Limits on CPU performance

Following is the result of performance of single core using High Performance Linpack (HPL). It illustrates variability among the different processors that could be assigned to a job. For example configuration 1 refers to Westmere-EP single core. Configuration 2 refers to AMD-Magny Cours (K-10), and the Henry2 configuration is what LSF assigns when a single core run is requested. Multi core runs may get different grade of processor including Westmere, Nehalem and other cores.

VCL-GP CPU configuration 1 (from Chapter 3):

Maximum measured single core HPL performance: 6.99 GFLOPS (10,000 order matrix, P=1, Q=1, NB=256)

ARC host-CPU configuration 2 (from Chapter 3):

Maximum measured single core HPL performance: 7.33 GFLOPS (16,000 order matrix, P=1, Q=1, NB=448)

VCL-HPC (Henry2) example CPU configuration:

CPU configuration for single core test: Processor Model – Intel Xeon, Clock Frequency – 2.8 GHz, Number of Processors – 2 (equals the total number of cores), No Hyper-threading, L1 cache – 16K data, L2 cache – 1024K Unified, Memory- 4 GB, CentOS release 5.4 (Final), X86_64 architecture.

Maximum measured single core HPL performance: 4.98 GFLOPS (16,000 order matrix, P=1, Q=1, NB=448)

A theoretical bound on processor performance:

A simple check on the performance can be made by calculating the number of flops one can obtain based on some very simple assumptions about the processor micro-architecture, for example the number of single precision floating point operations per second, ability to keep the pipeline full and CPU clock frequency.

A theoretical limit = (N = Number of cores) * (U = Number of floating point execution units / core) * (F = cycles / second or processor clock frequency) * (P = FLOP per cycle, i.e. the number of single precision floating point numbers per SSE register)

Example CPU configuration:

"Single core, one Floating point execution unit per core, 2 GHz clock frequency with 4 FLOP per cycle per Execution Unit or 128 bit SSE register"

A theoretical limit = (N = 1) * (U = 1) * (F = 2 GHz) * (P= 4)

A theoretical limit = 8 GFLOPS

The illustrated theoretical limit is a single precision limit. The double precision limit is almost half of the single precision but it also highly depends on the algorithm (mixed precision calculations for double precision numbers), compiler optimizations, vector SIMD operations, micro architecture optimizations, mix of different arithmetic operations in the code being executed (e.g.: % of multiplies, divides, adds and subtracts), etc.