A Network Architecture for Enabling Execution of MPI Applications on the Grid

Devaraj Das, Ritu Sabharwal, Sujoy Saraswati, P. N. Anantharaman,

Hewlett-Packard (STSD), Bangalore, India

Jason Oh

Hewlett-Packard, Singapore

{devaraj.das, ritu, sujoy.saraswati, anantharaman.pn2, jason.oh}@hp.com

Abstract

MPICH-G2 is a grid-enabled implementation of the MPI standard (version 1.1) for the Globus-2 device. MPICH-G2 requires point-to-point communication between the nodes where computations of a parallel application written using the MPI standard are scheduled. The point-to-point communication requires that all nodes where computations are scheduled be part of the same IP address space. This is a major issue when a node is actually a (Beowulf) cluster with a set of slave nodes in the private IP address space. In this paper, we propose a solution to this problem using the RSIP (Realm Specific-IP) framework.

Keywords: MPI, MPICH-G2, RSIP, RSAP-IP, Condor, Grid, Globus, RSL, GRAM

I. Introduction and Background

Grid computing, most simply stated is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing

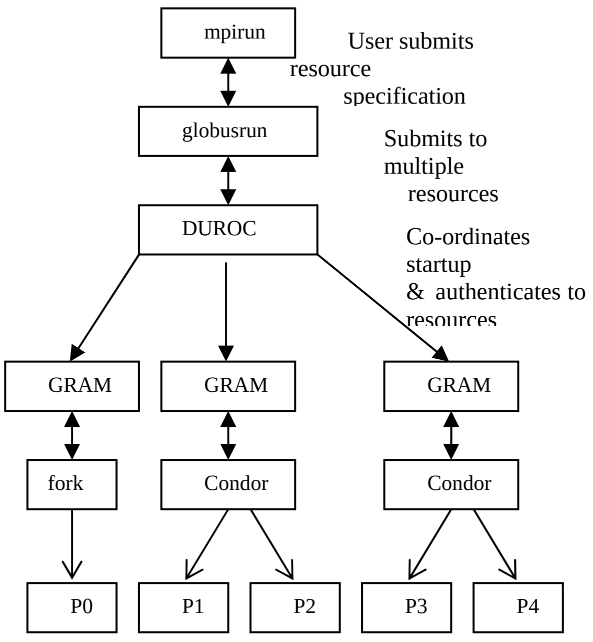

mpirun
User submits resource specification
globusrun
Submits to multiple resources
DUROC
Co-ordinates startup & authenticates to resources
GRAM
GRAM
GRAM
fork
Condor
Condor
P0
P1
P2
P3
P4

Figure 1. Parallel Application

various combinations of resources. There are no dedicated resources as in parallel computing; any resource available can be used for computation. Resources can be situated anywhere on the global network.

The Message Passing Interface (MPI) standard [1] provides a common interface for *message-based* communication and synchronization among processes executing on distributed-memory parallel computers and networks of workstations. MPICH [2] is an open-source implementation of the MPI standard widely used by a large community of Parallel Application developers. MPICH-G2 [3] is a *grid-enabled* implementation of the MPI v1.1 standard.

MPICH-G2 is based on services provided by the Globus Toolkit (*e.g.*, job startup, security) [4]. MPICH-G2 allows coupling of multiple machines, potentially of different architectures, to run MPI applications. MPICH-G2 automatically converts data in messages sent between machines of different architectures and supports multi-protocol communication by automatically selecting TCP for inter-machine messaging and (where available), vendor-supplied MPI for intra-machine messaging.

Figure 1 illustrates the steps by which a (MPI) parallel application is executed in a Globus grid system. As a first step, the user compiles the MPI application and links it against the MPICH-G2 libraries. He then identifies the resources (machines) on the grid, which are workstations and/or clusters (typically, *Beowulf* clusters [9]). He frames a Globus *Resource Specification Language* (RSL) [5] script and runs the grid-enabled *mpirun* command. An alternate here is to run the mpirun command with *machines* file as an input. The file contains a list of the resources. In this case, the RSL file is generated internally.

mpirun internally invokes *globusrun*, which parses the RSL script. globusrun uses the *Dynamically Updated Request Online Coallocator* (DUROC) [6] library to schedule the jobs on the resources (actually the jobs are sub-computations or sub-jobs of the single application). The DUROC library uses the *Grid Resource Allocation and Management* (GRAM) [7] protocol to request that a specific job be started on the resource. These resources typically have schedulers like Condor [8]. Once a resource receives a request for starting a job, it uses the requested *Job Manager* to launch the job. The Job Manager interacts with the scheduler that in turn schedules the job(s) on specific nodes under its control. In the case where resource is a cluster, the scheduler will schedule the computations on the compute nodes of the cluster. In the absence of a scheduler on the resource, default Job Managers like *fork* will be used to spawn the computation on the resource. (Note that fork here does not refer to Unix Operating System's fork system call). DUROC and the associated MPICH-G2 libraries manage the sub-computations of the single *large* computation.
The sub-computations communicate among themselves using TCP or vendor MPI (optimized

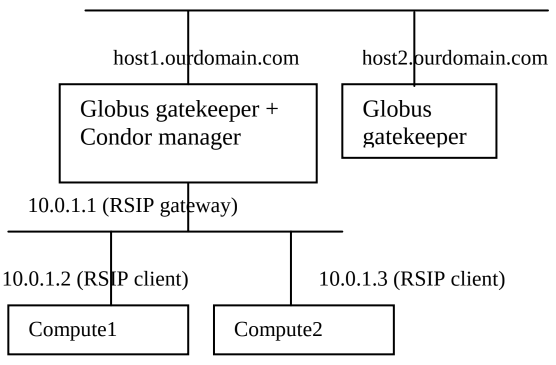


Figure 2. Setup of the test-bed

for the hardware). TCP is used for communication between the sub-computations if vendor MPI is not available. MPICH-G2 requires point-to-point communication between the nodes where jobs are scheduled. However, in case of Beowulf clusters, this requirement poses a major problem. The requirement essentially means that all the *compute* nodes of the cluster have to be in the global[1] IP address space. This is in conflict with a regular Beowulf cluster setup in which all the compute nodes are in a *private* IP address space (*e.g.*, 10.x.x.x network). We are constrained to launch MPI applications in a single cluster, even though the grid may host more than one cluster. This is an under-usage of available resources on the grid.

In this paper, we propose a solution to the problem described above. Our solution is based on the *Realm Specific IP* (RSIP) framework and protocol [10, 11]. We describe portions of the RSIP framework that are relevant to our work in a later section.

II. Problem Illustration

Figure 2 illustrates the setup that we used for deploying our solution. host1 is the head node of a (Beowulf) cluster with two compute nodes – Compute1 and Compute2. host2 is a standalone machine accessible directly to host1 on the main network. All nodes are Linux based and we do not have any vendor MPI in the cluster. The scheduler for the cluster is Condor. The RSL file shown in figure 3 can be used to submit an MPI application to be run on the setup. The submission can be done from host2 or from host1. In the setup, all machines are configured to be in the *ourdomain.com* domain.

count represents the number of computations (*sub-jobs*) of the application (under *executable* keyword) that is to be initiated on each resource. As was described earlier, *globusrun* parses the input RSL file and submits jobs to the resources (*resourceManagerContact*).

```
(&(resourceManagerContact="host1.ourdomain.com/jobmanager-condor")
   (count=2)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
          (LD_LIBRARY_PATH /usr/local/globus-2.4.3/lib/))
   (directory="/home/globus/test")
   (executable="/home/globus/test/cpi")
)
( &(resourceManagerContact="host2.ourdomain.com")
    (count=1)
    (label="subjob 2")
    (environment=(GLOBUS_DUROC_SUBJOB_INDEX 2)
                   (LD_LIBRARY_PATH /usr/local/globus-2.4.3/lib/))
    (directory="/home/globus/test")
    (executable="/home/globus/test/cpi")
)
```

Figure 3. A sample RSL file

The sub-jobs start the initialization process, and invoke a barrier that waits for all sub-jobs to reach a certain stage when they can run (*i.e.*, they are no longer stuck in some scheduler queue, etc.). The sub-jobs communicate with the Job Managers using *nexus* [14] messages that contain embedded IP addresses or DNS hostnames and ports. The Job Managers create hash tables using this information. The sub-jobs under each Job Manager use that table in

[1] Global with respect to the IP address of the machine where the application is launched. Global and *public* are used interchangeably in the paper, and they mean the same in the IP address context.

doing nexus communications with the hosts. After the nexus communications are successfully over, the sub-jobs are ready to run.

The sub-jobs then construct the data-structures that are used to identify their bind (IP address, port) point. This information, embedded in messages, is communicated to other nodes participating in the MPI application. This is later used for the MPI communication (*e.g.*, MPI_Send).

However, in the setup in figure 2, the barrier will not return successfully. This is because neither host2 can setup socket communication with Compute1 nor Compute1 can set up socket communication to host2 without special arrangements at the network layer. For similar reasons, the MPI communication cannot happen successfully.

In both the above cases, the implementation of Globus libraries and globus2 device of MPICH-G2 implementation is such that they use ephemeral ports. An ephemeral port is a port assigned by the OS to an application requesting it.
We will explain in the solution section how usage of ephemeral source ports helped us in devising the solution(s). We basically had to solve the problems of handling embedded hostnames in messages for the two cases described. In the next section, we discuss the RSIP protocol parts applicable to this work.

III. Realm Specific IP

RSIP is a new protocol and is an alternative to the Network Address Translation (NAT) [12]. NAT device sits between an internal network and the rest of the world. NAT device (router) peeks at each packet passing through it and modifies the IP header and/or the TCP/UDP header. The main issue with NAT is that end-to-end packet integrity is lost. Apart from this, NAT requires Application Layer Gateway (ALG) module in the router for each application protocol that embeds addressing information within the packet payload. RSIP addresses these limitations and removes them and is thus is a very good alternative to NAT.

RSIP is based on the concept of granting a host from one addressing realm a presence in another addressing realm. RSIP gateway would replace the NAT router. Figure 4 shows the high-level architecture of RSIP.

Hosts X (the RSIP client) and Y belong to different address spaces A and B respectively. Their IP addresses are Xa and Yb respectively. RG is a RSIP gateway with two interfaces whose addresses are Na and Nb respectively in the address spaces A and B. RSIP defines a set of protocol messages between the host and the gateway. When X wants to establish an end-to-end connection with Y, it sends a request to RG to assign it resources (resources here are IP addresses and ports). RG responds by assigning X resources and it creates a mapping of X's addressing information to the assigned resources.

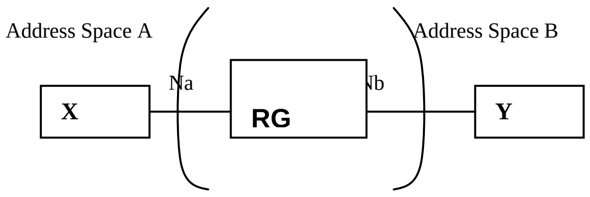


Figure 4. Architecture **of RSIP**

Hosts in the address space A use the assigned resources to tunnel data packets to RG. RG strips off the outer headers and routes the inner packets to address space B. The mapping created during the time of resource assignment is used to de-multiplex inbound packets from Y to X.

RSIP comes in two flavors – Realm Specific Address and Port IP (RSAP-IP) and Realm Specific Address IP (RSA-IP). We base our solution on the first flavor, *i.e.*, RSAP-IP.

A. RSAP-IP

In this case, each RSIP client is assigned the following:

- A port-range. This is used by the client applications as ephemeral source ports.
- An IP address. The IP address is the IP address of the gateway itself.

An RSIP client (for example, X in figure 2) creates a tunnel from itself to the RSIP gateway. The source IP address of the tunnel is set to the external IP address of the gateway (Nb), and the tunnel endpoint is set to Na. The routing table is set up such that the default route is through the tunnel interface. Due to the tunnel, hosts in the other side of the gateway (address space B in figure 2) get the impression that packets are being generated from RG and they respond as such. De-multiplexing of packets to X happens at RG as described earlier.

IV. Solution Overview

In the discussion that follows, address space A in figure 4 is referred to as the private IP address space and address B is referred to as the *public* IP address space. The compute nodes of a cluster are in address space A; RG is the head node of the cluster. RSIP server [13] is run on cluster head node and RSIP client on each of the compute nodes.

We have come up with two solutions to the problem. The first solution requires no changes to the source code (of Globus/MPICH) at all. It is based purely on the DNS service and RSAP-IP. The second solution requires changes to the Globus and the *globus2* device of the MPICH implementation.

A. DNS-based solution

In this solution, the requirement is that all nodes participating in the MPI application execution must have unique DNS names. DNS entries need to be setup for all compute nodes in the clusters and the FQDN (fully qualified domain name) of a compute node in a cluster must resolve to the corresponding head node's *public* IP address. For example, in figure 4, the DNS name Compute1 should resolve to 15.76.98.27. It is not practical to add entries for all compute nodes in the primary

DNS server of an organization. A more pragmatic alternative is to add entries in the DNS configuration-related files (such as /etc/hosts on Redhat Linux) of the hosts. Thus, in the setup that we have, the following DNS entries are added in /etc/hosts file on compute1 (and similar entries on compute2):

```
10.0.1.1 host1.ourdomain.com host1
10.0.1.2 compute1.ourdomain.com compute2
10.0.1.3 compute2.ourdomain.com compute3
15.76.98.218 host2.ourdomain.com host2
```

On host2, the following entries are made:

```
15.76.98.218 host2.ourdomain.com host2
15.76.98.27 compute1.ourdomain.com compute1
15.76.98.27 compute2.ourdomain.com compute2
```

Since the libraries use ephemeral ports, RSAP-IP can be effectively utilized on compute nodes. The OS on the compute nodes would always assign a port in the port-range assigned previously by the RSIP server. For example, if compute1 requests a port (by doing *htons(0)*) in the Globus/MPICH-G2 libraries, then the port assigned by the OS will be a port within the port-range assigned to compute1 by the RSIP server at the time of RSAP-IP registration. Thus, standard RSAP-IP protocol can be straightaway used in this case to de-multiplex incoming messages from remote nodes (like host2) to compute1. Also, compute1 (or compute2) can reach host2 through the tunnel route.

Though the above solution works for fixed number of clusters in a static environment, it lacks scalability since we have to create DNS entries for all participating nodes. We remove this limitation in the next solution; that solution is generic and scalable.

B. A general solution

The general solution is an extension of the DNS-based solution wherein we remove the necessity of having to modify files like /etc/hosts on the nodes. The first problem to tackle is to allow a successful invocation of barrier routine on each node. Whenever a nexus message is obtained on a socket *s*, the name of the peer connected to *s* is obtained (by doing *getpeername*). The nexus message is updated to have that name. The port-number values are not touched. The hash table creation process will take the modified embedded hostname as input. The peer name obtained from the socket will be necessarily different from the embedded hostname in cases where messages are sent from the compute nodes of a cluster to Job Managers on remote resources. This is because the hostname (actually IP address) in the IP packets would be the hostname of the respective head node. This effect is due to the usage of RSAP-IP protocol.

However, the important thing to note is that the above operations are performed on only the Job Manager side. Normal handling of nexus messages is done in cases (*i.e.,* embedded hostnames are used) where a compute node gets a nexus message from its corresponding Job Manager. The reason for this is that the compute nodes can communicate with remote resources directly (due to the presence of RSAP-IP). The assumption made here is that DNS names of the remote resources can be resolved by the compute nodes – for example, by setting the DNS server of the compute nodes to the head node and running DNS *relay* software on the head node.

Thus, the problems associated with the barrier are handled. As an example, consider a case where compute1 (with FQDN hostname compute1.ourdomain.com) sends a message to host2 with the hostname embedded in the nexus message. Host host2 will obtain the peer name of the other end of the connected socket and get 15.76.98.27. It will then use the DNS name corresponding to 15.76.98.27 (which is a well known name in the network) for the purpose of hashing, etc.

The next issue to be handled is the issue of embedded hostnames during the MPI application execution (after the initialization) so that communication using MPI primitives such as MPI_Send is enabled. An environment variable called MPICH_GLOBUS2_USE_NETWORK_INTERFACE, already defined in the MPICH-G2 libraries, is exploited in this case. The value of the variable is set in the RSL file to the IP address of the resource, which in case of a cluster is the IP address of the head node. Thus, in the compute nodes, the data-structure created to identify bind point uses the IP address of the head node. This IP address is also embedded in the message that is communicated to other nodes (both remote and peer compute nodes) participating in the MPI application; the latter use this IP address for talking to the node. This solves the problem of remote resources trying to communicate with any compute node in the cluster (since RSAP-IP will anyway do the de-multiplexing of incoming messages based on port numbers).

However, one complication arises because of the use of the environment variable. This issue has been discussed in section 6.7 of [10]. The RSIP client software creates a tunnel interface and the IP address of the tunnel interface is set to the IP address of the public side of the RSIP gateway. If two compute nodes in the same cluster wish to communicate (*e.g.*, using MPI_Send/Recv), and they both use the public IP address of the head node as a result of the environment variable being set, then this can result in a socket being set up between two RSIP hosts having the same source and destination IP addresses, which most TCP/IP stacks will consider as intra-host communication.

To handle the described situation, the RSIP server is made to save the port-mapping information to a well-known file whenever a compute node registers. The file is created in a file-system shared by all nodes in the cluster. For example, as per the setup illustrated in figure 2, a file with the following information[2] will be accessible to all nodes:

10.0.1.2	*8192*	*255*
10.0.1.3	*8447*	*255*

This means that 10.0.1.2 has been assigned 255 ports starting from 8192, and, 10.0.1.3 has been assigned 255 ports starting from 8447. This file will be looked up (information can be cached in local data-structures too) whenever a given compute node tries to communicate with a destination IP address same as the public IP address

[2] These are indicative values

of the head node. The input to the look-up will be the destination port number and this will return an IP address. Note that because the MPICH-G2 libraries use ephemeral ports this technique can be used. The destination IP address is then replaced with the IP address returned by the look-up and hence the communication will happen with the right compute node.

V. RSAP-IP overheads

RSAP-IP has some complications that have been discussed under section 6 in [10]. Some issues like *fragmentation*, *RSAP-IP protocol traffic*, etc. are potential complexities that can affect MPICH-G2 applications too. However, as discussed in [10], most of these issues can be mitigated. Issues like protocol traffic requirements can be minimized by setting large timeouts for recurring events like renewal of compute node registrations with the RSAP-IP server. We handle one specific complexity to do with *multi-party applications* by sharing the port-address mapping information with all nodes in the cluster. Some issues like *host deallocation* may not apply to MPICH-G2 applications at all. We are in the process of analyzing the other issues with RSAP-IP that can affect MPICH-G2.

VI. Related work

There were some efforts regarding the use of use of nexus proxy [15] but it is unclear whether the solution is applicable to the versions of Globus and MPICH-G2 available today [16]. The nexus proxy approach is based on SOCKS protocol. Since SOCKS protocol does not support the handling of passive open sockets, the solution requires *relay* processes to be run in the cluster and outside the cluster. In our approach, we do need the RSIP server and RSIP client processes, but if we carefully go through the set of steps that each of them does to set up tunnels, create mappings from assigned client ports to client IP addresses, etc., we will notice that we can do all that at system startup time and get rid of the daemon processes. [17] proposes a grid solution based on Java called Ibis and they address the problem of private cluster nodes communication with other external hosts using *relay* processes like [15]. Due to the presence of relays, both [15] and [17] require extensive code changes in MPICH-G2/Globus since the usual socket-related system calls cannot be directly used and performance is impacted as well. We solve the problem in a more simplistic, elegant, and, generic fashion by using RSIP.

VII. Conclusion and Results

In this paper, we have proposed two solutions to ameliorate the issue of MPICH-G2 requiring all nodes participating in an MPI application to be in the public IP address space. RSAP-IP is exploited for this purpose. The solutions work for real-world grid setup and are practical. The first DNS-based solution applies to small organizations where the number of clusters and their configurations in terms of number of compute nodes does not vary much over time. In case of the (more scalable) second solution, modifications to the Globus and MPICH-G2 libraries are minimal (tens of lines of code).

The solutions have been tested with a couple of MPICH-G2 applications in a variety of cluster deployments. Real-world MPI applications from Singapore MIT Alliance (SMA) were demonstrated on a grid setup at the LinuxWorld Conference, Singapore, May, 2004. The resources used were IA32 ProLiant Opteron Cluster of 4 blade servers, x86 running Linux and IA64 cluster of 2 nodes running Linux. The solutions will be used for the grid setup at the prestigious National Grid Office (NGO), Singapore (where they are unable to assign public IP addresses to compute nodes in clusters due to limited IP addresses). This work has also been submitted to Globus open source for review.

As part of future work, we will exploit the applicability of RSIP to solve firewall issues in running MPICH-G2 applications. We will also address the complications and interoperability issues (if any) in a network of clusters where some of the clusters have vendor-MPI. Performance studies of applications deployed in this architecture will also be done.

References

[1] MPI: A Message-Passing Interface Standard. http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html

[2] MPICH-A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich/

[3] MPICH-G2. http://www.niu.edu/mpi

[4] The Globus Alliance. http://www.globus.org

[5] The Globus Resource Specification Language RSL v1.0. http://www.globus.org/gram/rsl_spec1.html

[6] The Dynamically-Updated Request Online Coallocator (DUROC). http://www-fp.globus.org/duroc/

[7] Globus 2.4 Services http://www.globus.org/gt2.4/admin/guide-services.html#gram

[8] Condor Project. http://www.cs.wisc.edu/condor/

[9] Beowulf Cluster. http://www.beowulf.org/

[10] M. Borella, et al. "RFC 3102 - Realm Specific IP: Framework", October 2001. http://www.faqs.org/rfcs/rfc3102.html

[11] M. Borella, et al. "RFC 3103 - Realm Specific IP: Protocol Specification", October 2001. http://www.faqs.org/rfcs/rfc3103.html

[12] Network Address Translation. http://www.vicomsoft.com/knowledge/reference/nat.html

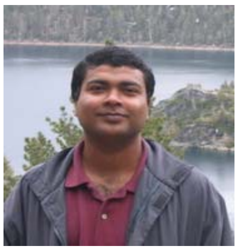
[13] RSIP for Linux. http://openresources.info.ucl.ac.be/rsip/

[14] The Nexus Multithreaded Communication Library. http://www.globus.org/nexus/

[15] Mark Baker, et al. "A Report on Experiences Operating the Globus Toolkit through a Firewall". http://esc.dl.ac.uk/Papers/firewalls/globus-firewall-experiences.pdf

[16] Mailing-list discussion. http://www-unix.globus.org/mail_archive/mpich-g/2004/03/msg00012.html

[17] Alexandre Denis, et al. "Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems", http://hpdc13.cs.ucsb.edu/papers/33.pdf



Devaraj Das is a post graduate in Computer Science from Indian Institute of Science, Bangalore, India. He has been working with Hewlett-Packard ISO, Bangalore since July 1997. His interests are in the area of Distributed Systems, Networking and Security technologies. He has actively contributed in the deployment of Grid-based solutions in multiple customer environments.

Ritu Sabharwal is a post graduate in Computer Science from Indian Institute of Science, Bangalore, India. She has been working with Hewlett-Packard ISO, Bangalore since February 2003. In her role as a Senior Systems/Software Engineer at HP she has been working on the SmartFrog project in collaboration with HP Labs, Bristol, UK and has also actively participated in Research and Development of Grid related technologies and solutions.

Sujoy Saraswati is a post graduate in Computer Science from Indian Institute of Technology, Kharagpur, India. He has been working with Hewlett-Packard ISO, Bangalore from last five years. In his role as a Software Engineer in HP he has been working in the Caliper project under the STSD division. HP Caliper is a powerful performance measurement tool for Itanium applications. Sujoy has also been associated with the work for enabling execution of MPI applications on the grid.

Anantharaman (Ananth) manages the Grid program, compiler, developer tools and Java products at Hewlett Packard, India Software Operations. The key areas of work in his group include developing toolsets for grid deployment, virtualization technologies and compiler/java technology for PA RISC and Itanium platform. His group is actively engaged in several projects for NGPP. Ananth has an extensive industry experience of over 20 years as a manager as well as a technologist working in systems technology. Ananth holds a masters degree in Electronics from Indian Institute Of Science, Bangalore, India.

Jason Oh is a post graduate in Computing and Information Systems from University of London, UK. He has been working with Hewlett-Packard Singapore since July 2000. In his role as a Solution Architect in HP, he provides consultation and architect solutions for High Performance Computing (HPC) and Grid requirements.