# The PRAGMA Testbed – Building a Multi-Application International Grid

Cindy Zheng, David Abramson, Peter Arzberger, Shahaan Ayyub, Colin Enticott, Slavisa Garic, Mason J. Katz, Jae-Hyuck Kwak, Bu Sung Lee, Phil M. Papadopoulos, Sugree Phatanapherom, Somsak Sriprayoonsakul, Yoshio Tanaka, Yusuke Tanimura, Osamu Tatebe, Putchong Uthayopas

## Abstract

*This practices and experience paper describes the coordination, design, implementation, availability, and performance of the Pacific Rim Applications and Grid Middleware Assembly (PRAGMA) Grid Testbed. Applications in high-energy physics, genome annotation, quantum computational chemistry, wildfire simulation, and protein sequence alignment have driven the middleware requirements, and the testbed provides a mechanism for international users to share software beyond the essential, de facto standard Globus core. In this paper, we describe how human factors, resource availability and performance issues have affected the middleware, applications and the testbed design. We also describe how middleware components in grid monitoring, grid accounting, grid Remote Procedure Calls, grid-aware file systems, and grid-based optimization have dealt with some of the major characteristics of our testbed. We also briefly describe a number of mechanisms that we have employed to make software more easily available to testbed administrators.*

This work was supported in part by the U.S. National Science Foundation under Awards INT-0314015 and SCI-0505520; by the Ministry of Education, Sports, Culture, Science and Technology of Japan through the National Research Grid Initiative Project; by Australian Government under a variety of awards including the Cooperative Research Centre scheme, the Departments of Education Science and Technology and Communications, Information Technology and the Arts, the Australian Research Council and Monash University; by the Thailand Research Fund and Kasetsart University Research and Development Institute SRU fund; by a grant from the Ministry of Information and Communication through the K*Grid project; and by National Grid Office of Singapore.

C. Zheng, P. M. Papadopoulos, and M. J. Katz are with San Diego Supercomputer Center, USA (zhengc@sdsc.edu); D. Abramson, S. Ayyub, C. Enticott, S. Garic are with Monash University, Australia (david.abramson@infotech.monash.edu.au); P. Arzberger is with University of California, San Diego, USA (parzberg@ucsd.edu); J. Kwak is with Korea Institute of Science and Technology Information, Korea (jhkwak@kisti.re.kr); B. S. Lee is with Nanyang Technological University, Singapore (EBSLEE@ntu.edu.sg); Y. Tanaka, Y. Tanimura and O. Tatebe are with National Institute of Advanced Industrial Science and Technology, Japan (yoshio.tanaka@aist.go.jp); P. Uthayopas, S. Phatanapherom and S. Sriprayoonsakul are with Kasetsart University, Thailand (putchonguth@hotmail.com).

## 1. Introduction

The Pacific Rim Applications and Grid Middleware Assembly (PRAGMA), founded in 2002, is an open international organization that focuses on a variety of practical issues of building international scientific collaborations in a number of application areas. PRAGMA operates four working groups – resources, telescience, bioscience and data computing. The resources working group's mission is to improve the interoperability of grid middleware and to enhance the usability and productivity of a global grid.

In May 2004 we established an experimental global grid testbed and ran multiple applications on a routine-basis. This experiment allowed us to learn about issues involved in building an easy-to-use production global grid. The testbed itself is a "grass roots" effort to build a long-term working experiment that allows a variety of applications and application scientists to work across international boundaries.

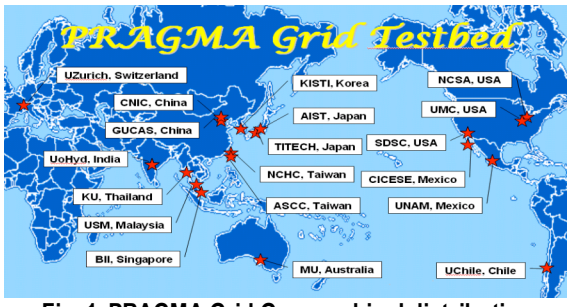


**Fig. 1. PRAGMA Grid Geographical distribution**

Today, 19 institutions from 5 continents and 13 countries have contributed both physical resources and, critical to success, people resources to the testbed. It is an instantiation of a useful, interoperable, and consistently available grid system that is neither dictated by the needs of a single science domain, nor funded by a single national agency. It differs

significantly from experiments such as PlanetLab [1] and TeraGrid [2] in that it does not have uniform infrastructure management. We started with minimum testbed system requirements: a local job scheduler and Globus [3]. Upper middleware layers and components are gradually built-in, driven by applications requirements and grid management needs.

Over 18 months we ran 5 applications - Time Dependent Density Functional Theory (TDDFT [4]), Quantum Mechanics / Molecular Dynamics (QM/MD [5]), the CSIRO Conformal-Cubic Atmospheric Model (CCAM [6]), a genetic sequence alignment tool (mpiBLAST [7]), and the integrated Genome Analysis Pipeline (iGAP [8]). Each application creates its own sub-grid and middleware stack: TDDFT and QM/MD are based on Ninf-G [9], CCAM on Nimrod [10], mpiBLAST on Mpich-g2 [11], and iGAP on Gfarm [12] (Fig. 2). Ninf-G, Nimrod and Mpich-g2 are application middleware which enable applications to run on grid.

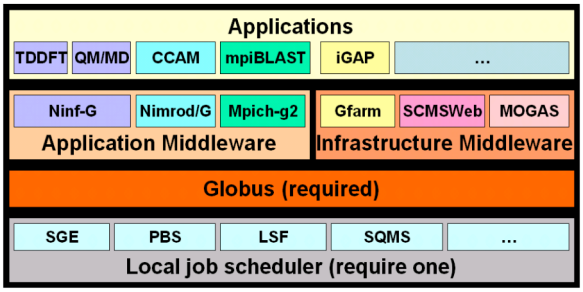


**Fig. 2. PRAGMA Grid Software Layers**

Simultaneously, we built the grid infrastructure by establishing a grid operation center, deploying a grid monitoring system (SCMSWeb [13]), testing a grid accounting system (MOGAS [14]) and building a grid file system (Gfarm). SCMSWeb, MOGAS and Gfarm are infrastructure middleware which provide and enhance the grid infrastructure services, thus benefit applications and grid management.

Throughout our experiments we encountered many problems, including system/network failures, trust management issues, complications in user and application environment setup, job submission difficulties, resources sharing, system/job monitoring and accounting problems. These problems were mainly caused by four factors: distance and time zone differences among sites, lack of infrastructure tools for heterogeneous global grid, non-uniform system and network environments, and diverse application requirements. Addressing these issues and problems has focused our efforts, motivated collaborative innovations and accelerated software enhancements

## 2. The Testbed

Mobilizing human resources was the first and most significant issue in building the testbed, which created some unique challenges because of the geographic distribution or our testbed sites. Furthermore, each site had its own funding sources and chain of command, thus it was not always easy to allocate the human resources, or achieve the required priority, to this project. Out of necessity, we based our relationship on mutual benefits and interests, and built consensus through active leadership. We formed a testbed working team by enlisting two contacts per site: a technical contact and a supervisor. A team member acted as the coordinator who maintained close communication between the team members and the PRAGMA leadership. The coordinator was responsible for soliciting and encouraging input from the team, organizing discussions, summarizing team decisions, tracking task status and promoting speedy progress in implementing the team decisions.

The main communication tool used by the testbed support team was email. We established mailing lists for both supporters and users. In addition, video teleconference and audio teleconference via Skype [15] were also used for group meetings. Whilst effective, these real-time communication methods were difficult to organize and chair, since they typically involved 10 people from five groups distributed across multiple time zones (between 10 and 14 hour separations). Moreover, whilst we all speak English, it is not the native tongue for the majority sites. Over and above these electronic meetings, PRAGMA semi-annual meetings proved to be precious opportunities and allowed most sites get together for face-to-face meetings.

A key instrument for organizing our activities was the Grid Operation Center (GOC). Via the GOC we compiled and published a testbed member list, resource tables, application requirements, testbed support status and user guides. We installed the SCMSWeb meta-server (§ 4.2) to provide real-time testbed status on both systems and jobs. Further, Asian Pacific Area Network and National Laboratory for Applied Network Research provided real-time network status and measurements on GOC. Finally, the GOC was used to archive and publish research data, reports, software and conference presentations.

Heterogeneity is a distinct characteristic of a grass roots grid. In our case, each site had its own policies, which precluded standardizing system environments, policies, procedures among testbed sites. Each site

specified their own software versions, security policies and maintenance schedules, and we had to tolerate these changes and try to adapt with better technologies and solutions. This is in stark contrast to the US TeraGrid in which a standard software stack is used for all grid sites.

With no well-established methods on how to conduct our experiments, we developed the following (still evolving) procedure:

1. Since the testbed is built according to application requirements, to prepare an application run, the coordinator first asks the application driver to document application requirements and deployment instructions. The coordinator also provides GOC access for the application drivers to publish the documents on the GOC.
2. The coordinator implements the requirements locally and works with the application driver to resolve any problems and to simplify, refine deployment instructions in order to minimize work load for all testbed site administrators.
3. The coordinator emails the testbed supporters list, calls for general deployment of the requirements.
4. When a site has implemented the requirements, the site technical contact emails both the application driver and the coordinator. The application driver then tests implementation at that site and resolves problems directly with the site technical contact.
5. When the application is working correctly at the site, the application driver includes the site in his/her routine long runs. The coordinator then updates the site support status table on the GOC website.

The first thing bonds different sites into a grid is trust. Our PRAGMA Grid Testbed employs GSI (Grid Security Infrastructure)-based authentication and authorization. Each institution runs its own Certificate Authority (CA) and issues certificates for users, hosts, and services which belong to the institution. At the beginning of building our testbed, all institutions agreed to trust each other's CAs. Although most CAs were experimental, this decision was accepted within the PRAGMA institutions to accelerate the development of the testbed. Last year, many CAs were upgraded to production-level and are expected to be trusted by other Grid communities such as TeraGrid and EGEE. The coordination of the security policy is managed by the Asia Pacific Policy Management Authority (APGrid PMA) and the International Grid Trust Federation (IGTF). As of Feb. 2006, our testbed has five IGTF-accredited CAs which have been used for multi-Grid interoperation. In addition, PRAGMA is planning to launch a PRAGMA CA which is expected to be an IGTF-accredited catch-all CA for PRAGMA members.

## 3. Application Middleware

The PRAGMA testbed is built on top of a Globus core, and this provides all of the standard low level services. Over and above this we have implemented an upper middleware layer consisting of tools, such as Nimrod/G and Ninf-G, more closely related to the applications. Whilst Nimrod and Ninf-G support very different programming methodologies, they both had to handle a number of similar issues that arose in the testbed, namely fault tolerance, application deployment and setup, and resource configuration. Specifically, Ninf-G implements GridRPC, and thus application programs consist of a set of functions, some of which are invoked by a local procedure call, and some of which are invoked across the network by a remote procedure call. Nimrod/G, on the other hand, runs complete applications on a single grid resource, but achieves parallelism by executing more than one application at a time (although, Nimrod can run parallel applications as well).

### 3.1. Fault Tolerance

Faults are normal in the operation of the testbed. Any resource and network connections can fail at any time. In a RPC based system like Ninf-G, a failure in a remote procedure would normally cause the calling client code to hang. However, Ninf-G monitors the remote procedure, and if a failure occurs, the return code of the call is set to error status, and the call fails accordingly. It is then up to the application programmer to insert code to handle the failure – possibly by causing the procedure to be retried elsewhere. A remote failure is detected by watching status of the TCP connection between the client process and the remote process. Ninf-G allows the programmer to set a timeout parameter on the call, and this prevents the application from hanging if no response occurs. In our routine-basis experiment, we found the timeout mechanism particularly helpful when the network was unstable. In addition, the experimental results guided us to improve the timeout mechanism so that Ninf-G uniquely detected three types of timeout: job start, job terminate, and session. The routine-basis experiment provided Ninf-G developers with opportunities to improve Ninf-G at the points of stability and fault detection, which are not easy to test on a single cluster or a national-level grid

interconnected via a high speed network.

Nimrod/G was equally affected by network and processor faults. Because Nimrod only runs whole programs, a failure normally means that the entire program execution is lost. Accordingly, Nimrod/G includes a retry heuristic that launches the application again in the event of an individual job failure. The job status is reported back to the user through the job monitor, and a user can decide whether the job should be retried or not. To date, this simple retry technique has been powerful enough for all the applications we have run. However, the climate experiment used in this study required us to execute 90 independent model runs, where each one actually ran for a number of weeks! In this situation, a single job failure would stall the results whilst the single run was repeated, and this would add an unacceptable delay.

To solve this problem, we modified the Nimrod/G scheduler to support parallel parameters (Parameters) and sequential ones (Seqameters). These can be used to add an additional seqameter, called `time_step`, to the experiment; thus, Nimrod/G runs each of the 90 simulations for each time step before moving onto the next time step. At the end of each time step, the results are copied back to the root node controlling the experiment, and are also left on the scratch storage of the machine that performed the work. When the next time step starts, Nimrod/G copies these output files as input to the next stage – either from the root node or the compute node, depending on which one has better connectivity to the downstream computation. In some circumstances, the scheduler will place the dependent computation on the same node as its predecessor, removing the need to copy the files altogether. In the event of a failure, the standard Nimrod/G retry mechanism is used, but for only one time step, and thus the losses are much smaller. Of course, it is possible to make the time steps quite small, at the cost of increased network copies for the input and output files.

### 3.2. Application deployment and setup

Both Ninf-G and Nimrod/G rely on executable code being available on a target system. There are two choices for application deployment. One is manual installation, in which the user logs into all remote computers, and installs the applications manually. The advantage is that users can configure the application for the machine architecture and its configuration. The disadvantage is that the work is tedious, repetitive and error prone may leads to version mismatches. The other method is automatic distribution by the staging function in Ninf-G or the remote copy function in Nimrod/G. When users specify the executable to be shipped to the remote in the configuration file of the client, the executable will be transferred before it is spawned on the remote resource. But, this method requires the user to prepare the executable for all remote architectures and to make them available on the root machine.

Ultimately the choice depends on how often the remote executable will be modified and how many target architectures there are. If the remote calculation is stable and does not need frequent modification, automatic distribution would be advantageous, especially if target systems are relatively homogeneous. Similarly, application setup can be done manually or automatically. This phase can also be time consuming. For example, in the Nimrod/G climate study we need to distribute 250 MByte files to each `time_step` – a total of 80 GBytes. In this instance, we chose to ship all files to all resources, giving the scheduler the flexibility to place the computation on any resource. In this case, distributing the file by GridFTP or FEDEX takes about the same amount of time (4 days per site), but the network was simpler to set up and control. The alternative was to copy each 250 MByte file automatically to the remote resource only when required, however, this adds a substantial startup cost to the computation and therefore was less desirable than just making the entire data set available. We hope to use Gfarm (§ 4) to resolve this type of problem in the future.

### 3.3. Configuration of resources

Both Ninf-G and Nimrod/G are designed to handle resource heterogeneity. For example, in Ninf-G, the first argument of a client program must be a “client configuration file’’, in which information required for running application is described. In order to compensate for the heterogeneity and unreliability of the grid, Ninf-G provides client configuration formats for detailed description of server attributes such as the port number of the Globus gatekeeper, the local scheduler type and its queue name for submitting jobs, a protocol for data transfers, library path for dynamic linking, etc.

The detailed client configuration format enables Ninf-G applications to adapt to heterogeneous configuration of clusters. Similar techniques are available in Nimrod/G; however, the experiment performed here did require some changes that we had not predicted. For example, different file system

structures required some minor modifications to Nimrod to run on clusters that did not have a shared file system across the nodes.

## 4. Infrastructure Middleware

### 4.1. Gfarm

Gfarm is infrastructure software we deployed and tested with iGAP application in PRAGMA testbed. As a grid file system, Gfarm facilitates reliable file sharing and high-performance data computing across different administrative domains. It is a scalable virtual file system federating local file systems of cluster nodes. Efficient utilization of local file system of compute nodes is a key technology for scalable parallel and distributed data computing. Another key feature is the management of file replicas in file system metadata for fault tolerance and load balancing. Furthermore, Gfarm syscall hook library and GfarmFS-FUSE enable existing applications access a Gfarm global virtual file system without any modification. Gfarm syscall hook library emulates of mounting Gfarm file system by trapping system calls for files in applications, where as GfarmFS-FUSE physically mounts a Gfarm file system in user space via FUSE mechanism. This feature makes Gfarm file system a fundamental infrastructure of a shared file system for grid.

**4.1.1. Application deployment.** Gfarm is a network shared file system for grids that supports binary execution and shared library loading. Once an application is installed, and necessary input data is untarred in Gfarm file system, it can be executed on any cluster node in grid. No explicit staging of applications and input data is required. Moreover, different kinds of binaries can be stored at the same pathname in Gfarm file system. When two kinds of binaries for Linux and Solaris are installed at /gfarm/bin/igap, appropriate binary is automatically selected and executed depending on the platform architecture. This feature helps to mitigate troublesome application deployment and setup in a heterogeneous environment.

**4.1.2. Performance.** During an installation and setup process of iGAP, genome annotation software, we encountered a performance problem. Gfarm has been developed to improve reading and writing performance of large files. In this case the total I/O throughput is more critical than fast access to metadata. However, the installation and setup process requires copying or untarring thousands of files including programs and input data. In this case, the metadata access overhead is crucial to performance. To reduce the overhead, we have improved the metadata cache management, and introduced a metadata cache server to share the metadata cache among different applications. As a result, the performance improved dramatically. For example, directory listing of 16,393 files took 44.0 seconds. Improving the metadata cache management reduced this to 3.54 seconds. The introduction of a metadata cache server at the client further reduced this to 1.69 seconds.

***4.1.3.*** **Fault tolerance.** Gfarm file system supports fault tolerance in case of disk and network failure by automatic file replica selection in access time. On the other hand, this feature works only when there is an available file replica somewhere. To ensure this, automatic file replica creation mechanism is required. Since Gfarm implementation in PRAGMA testbed is a worldwide distributed file system, there are several requirements for the mechanism. One is to have at least two file replicas for each file in case of disk failure, and another is at least one file replica for each file at each site for performance and fault-tolerance. We developed file replica creation command to meet the requirements. Currently, this command is regularly executed by cron.

### 4.2. SCMSWeb

SCMSWeb was the first infrastructure software deployed in PRAGMA testbed. It serves as the eyes to the testbed, which in turn it helped SCMSWeb to discover problems and improve in stability, scalability and performance. SCMSWeb monitors and reports a wide variety of system usage and performance metrics (such as load average, CPU/memory usage, disk/network I/O activity, page/swap/context switching rate, system temperature) as shown in Fig. 3.

***4.2.1.*** **Heterogeneity.** The testbed heterogeneity arises from the diversity of site physical locations, network conditions, hardware architectures, operating systems (e.g. Linux, Solaris), software stacks and versions etc. This characteristic provides a realistic global grid environment and challenges to our software and also provides a collaborative environment for software improvements. For example, prior to testing in testbed, SCMSWeb supported limited platforms. Whence SCMSWeb being deployed in the testbed, a Solaris site at Centro de Investigacion Cientifica y de Educacion Superior deEnsenada (CICESE), and an IA64 site at Computer Network Information Center (CNIC), volunteered to help porting SCMSWeb. With

access to these platforms and help at these sites, we were able to port SCMSWeb to Solaris and IA64 in only a few weeks. Further, as SCMSWeb was deployed in the testbed, we received a lot of feedback from sites, including problem reports, new feature and interface change requests. These valuable input stimulated fast improvements in software reliability and usability.


**Fig. 3. Grid Meta View and System Information View**

The testbed heterogeneity also provides technical challenges in information exchange and storage. Driven by the demand of PRAGMA test-bed, the SCMSWeb team decided to use an XML format to describe most of the information processed, exchanged, and stored. The main advantage of XML is portability. We also found that using XML allowed us to easily extend the format of information to accommodate new features, while maintaining the compatibility between each version of software.

**4.2.2. Reliability.** Another major issue presented by the testbed was reliability of the grid. This arose from the large distances between resources and the different levels of networking infrastructure in each country. Our experiences show that most grid reliability problems were caused by either network failure or poorly configured Globus software. To monitor such problems, a set of probes were developed and added to SCMSWeb to test network and Globus services. This feature helps site administrators to detect and fix problems in a more timely fashion, substantially improved the reliability and productivity of the grid. An example of probes view is shown in Fig. 4.

**4.2.3. Software Deployment.** Software deployment in large-scale grid is also a challenging problem. Currently, software distribution is done using the Linux RPM format. However, dependency can be a problem on for some Linux distributions. So, we adopted the solution to regenerate the RPM source file (.src.rpm) and to install this source on the system. On systems such as Solaris, the desired format is source code packed as a compressed tar file. The installation order is also a concern, so, care must be taken to preinstall required packages before compiling SCMSWeb. For newly installed clusters, users can install the OpenSCE [16] Roll that comes with Rocks [17], which includes a preinstalled and pre-configured version of SCMSWeb. This makes SCMSWeb installation and configuration very easy. With both the source distribution method and the roll distribution method, we have managed to reduce the installation effort while still cover wide range of demands arose within a heterogeneous testbed.

::: PRAGMA Status:::

| Site | Summary | Authentication (0/0/20) | DNS (18/0/2) | GridFTPFr (0/5/15) | GridFTPTo (0/6/14) | JobRun (0/0/20) | Mds (7/7/6 |
|---|---|---|---|---|---|---|---|
| amata1.cpe.ku.ac.th | (2/0/4) | | | | | | |
| ase.nchc.org.tw | (2/0/4) | | | | | | |
| malicia.super.unam.mx | (2/0/4) | | | | | | |
| marlin.bii.a-star.edu.sg | (2/0/4) | | | | | | |
| pragma001.grid.sinica.edu.tw | (2/0/4) | | | | | | |
| solaris-1.cicese.mx | (2/0/4) | | | | | | |
| ume.hpcc.jp | (2/0/4) | | | | | | |
| gsic-presto.titech.hpcc.jp | (1/3/2) | | | | | | |
| hawk.usm.my | (1/3/2) | | | | | | |
| aurora.cs.usm.my | (1/2/3) | | | | | | |
| rocks-52.sdsc.edu | (1/2/3) | | | | | | |

**Fig. 4. Probe view show grid service status**

**4.2.4. Performance.** Initially, SCMSWeb was developed as a cluster monitoring system. As the number of sites in PRAGMA testbed increased, there was a need to enhance both scalability and performance of SCMSWeb. To handle the scalability requirement of the testbed, we enhanced SCMSWeb to support a hierarchical monitoring structure. In this structure, a higher level node can easily pull system status information from lower level systems. The topology of monitoring tree depends on the agreement among the participating sites. Currently, the root node of the monitoring tree for PRAGMA grid is located at http://pragma-goc.rocksclusters.org/scmsweb. With this enhancement, a system administrator can easily get most of the information needed to understand the grid operational characteristic from root node.

When network bandwidth is low, we found that the transfer of large monitoring information tends to be delayed or fails. This causes some information to be loss or not updated. To avoid this problem, we improved SCMSWeb by adding data compression during the transfer process. Since the original data is text based XML data, we successfully reduced the size of data being transferred up to 10 times. As a result, the

performance and stability has been improved substantially.

### 4.3. MOGAS

Multiple Organization Grid Accounting System (MOGAS), developed by Nanyang Technological University in Singapore, has been deployed to 7 sites in PRAGMA testbed since May 2005.

MOGAS was developed to log the job submission across multi-organization clusters. It is built on top on of Globus core: Globus Resource Allocation Manager (GRAM), Globus Security Infrastructure (GSI) and grid file transfer protocol (GridFTP). MOGAS supports a number of well known local cluster schedulers. The PRAGMA testbed provides a good, heterogeneous environment and diverse set of job schedulers (various flavors of PBS, LSF, and SGE) for MOGAS testing. Working in PRAGMA testbed, we found that different GRAM scripts and different security policies among testbed sites required us to improve our deployment procedure, automate and take into account all of the various requirements, and ease its installation and setup. Also, after the MOGAS deployment in the PRAGMA testbed, we received significant feedback and suggestions, and these resulted in immediate improvements and provided us with clear directions for future enhancements.

Some of the features available in MOGAS include: overall consumer/provider usage view in a table matrix form (job and runtime); resource trading ratio view; resource workload view; the top 10 resource providers and consumers; and Automatic account report generation. (http://ntu-cg.ntu.edu.sg/pragma)

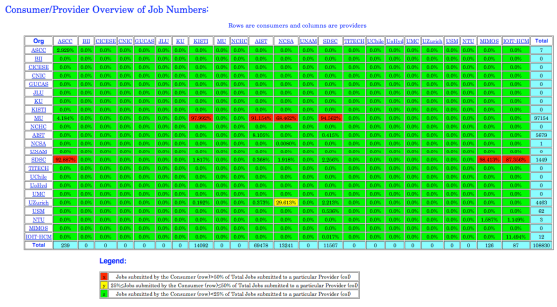


**Fig. 5. MOGAS Consumer/Provider matrix table**

Fig. 5. shows the consumer/provider table matrix. It shows the percentage of job submitted and the total number of jobs submitted by each organization in last column in the table.

An analysis of the log data shows that there is a high amount of inter-organization resource usage and the test-bed is quite stable, with some jobs running successfully for more than 50 hours. We also observe the job type and characteristics using the different middleware and fine tune the middleware and application to make effective use of the testbed.

## 5. Conclusions

In this paper we discussed the PRAGMA testbed and some key lessons we learned during our routine use experiments. The testbed has these characteristics:

- Faults (hardware, software and network) are normal and must be handled both by middleware and in some cases, applications themselves.
- The testbed concurrently supports a rich variety of application needs, including high performance computation, high bandwidth data access.
- The testbed is large, allowing applications to explore issues that would need to be dealt with at massive scales such as, resource availability, network bandwidth and contention, distributed resource monitoring, and global name spaces.
- Coordination of resources, monitoring, inter-operable software deployments, and resource/user policies are significantly complicated because applications may have different requirements for middleware and resource allocation. Both application users and system administrators are located across 12 time zones. This presents a challenge in putting together a team with very diverse cultural backgrounds.
- The testbed covers many countries with different levels of network speed, quality, and system resources. Thus, the situation that middleware has to deal with is more realistic, and more challenging than grids that are built only in countries with strong infrastructure in place. New types of middleware can be deployed and tested in a wide variety of administrative and physical environments.
- A telling value of the testbed is how its use created strong feedback between application and middleware (e.g. Ninf-G, Nimrod and Gfarm). Software had to be improved, then deployed quickly (e.g. SCMSWeb, MOGAS). Finally, collaboration among testbed sites helped to improve the software (e.g. porting SCMSWeb to Solaris and ia64).

Applications in variety of fields have driven the middleware requirements and the testbed provides a mechanism for users around the world to share and use software that goes beyond the essential, *de facto* standard Globus core. We've found that at the initial

stage, by setting absolute minimum grid-wide software requirements (Globus and a local scheduler) with additional middleware dependent on application needs, deployment across all institutions in a grass-roots global grid is more realistic and manageable. But the infrastructure middleware components could become part of the required software infrastructure in the future. (Fig. 2.)

Given daunting issues such as lack of centralized control and routine system faults, it would seem that such a system would simply be unusable. However, our experience has shown just the opposite. Real science can be performed on such a system, and the environmental challenges greatly improve the middleware systems. We hope that our experience will inform other emerging grid efforts in operations of their grids, and concurrently will focus research development on the practical needs of these grids.

## 6. Acknowledgments

The authors wish to acknowledge the support of all members of the PRAGMA Resources Working Group and of the PRAGMA testbed, without whose support, the results presented in this paper could not have been achieved. These include Academia Sinica Computing Center, Taiwan; Bioinformatics Institute, Singapore; Center of Investigation Science and Education at Superior Ensenada, Mexico; Computer Network Information Center, Chinese Academy of Science, China; Kasetsart University, Thailand; Korea Institute of Science and Technology Information, Korea; Monash University, Australia; National Center for High-performance Computing, Taiwan; National Grid Office of Singapore; National Institute of Advanced Industrial Science and Technology, Japan; National Center for Supercomputing Applications, USA; National University of Mexico, Mexico; San Diego Supercomputer Center, USA; Tokyo Institute of Technology, Japan; University of Hyderabad, India; University of Sains Malaysia, Malaysia.

## 7. References

[1] Larry Paterson and Timothy Roscoe, "The Design Principles of PlanetLab", PlanetLab Consortium, PDN--04--021, June, 2004.

[2] Rob Pennington, "Terascale Cluster and the TeraGrid", Proc. of HPC Asia 2002, pp. 407--413, 2002.

[3] Ian Foster and Carl Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputer Applications, v11, #2, pp. 115--128, 1997.

[4] K. Yabana and G. F. Bertsch, "Time-Dependent Local-Density Approximation in Real Time: Application to Conjugated Molecules", Quantum Chemistry, Vol. 75, pp. 55-66, 1999.

[5] QM/MD, http://www.globusworld.org/program/abstract.php?id=64.

[6] McGregor, J.L., Walsh, K.J. and Katzfey, J.J. Nested modelling for regional climate studies. In: A.J. Jakeman, M.B. Beck and M.J. McAleer (eds.), Modelling Change in Environmental Systems, J. Wiley and Sons, 367–386, 1993.

[7] A. Darling, L. Carey and W. Feng , "The Design, Implementation, and Evaluation of mpiBLAST", *4th International Conference on Linux Clusters: The HPC Revolution*, San Jose, CA, June 2003.

[8] Wilfred W. Li, Peter W. Arzberger, Chang Lim Yeo, Larry Arg, Osamu Tatebe, Satoshi Sekicughi, Karpjoo Jeong, Suntae Hwang, Susumu Date, Jae-Hyuck Kwak, "Proteome Analysis using iGAP in Gfarm", Proceedings of 2nd International Workshop on Life Science Grid (LSGRID 2005), 2005.

[9] Yoshio Tanaka, Hiroshi Takemiya, Hidemoto Nakada, and Satoshi Sekiguchi, "Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid", Fifth IEEE/ACM International Workshop on Grid Computing, pp. 298-305, Nov. 2004.

[10] Abramson, D, Buuya, R. and Giddy, J. "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker", Future Generation Computer Systems. Volume 18, Issue 8, Oct-2002.

[11] Foster, I. and N. Karonis. A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems.inProc. SuperComputing 98 (SC98). 1998. Orlando, FL: IEEE.

[12] Osamu Tatebe, Noriyuki Soda, Youhei Morita, Satoshi Matsuoka, Satoshi Sekiguchi, "Gfarm v2: A Grid file system that supports high-performance distributed and parallel data computing," Proceedings of the 2004 Computing in High Energy and Nuclear Physics (CHEP04), Interlaken, Switzerland, September 2004.

[13] Napat Chalakornkosol and Putchong Uthayopas, "Monitoring the Dynamics of Grid Environment using Grid Observer ", Poster Presentation in IEEE CCGRID2003, Toshi Center, Tokyo, May 12-15, 2003.

[14] Bu-Sung Lee, Ming Tang, Junwei Zhang, Ong Yes Soon, Cindy Zheng, Peter Arzberger, David Abramson "Analysis of Job in a Multi-Organizational Grid Test-bed", International Workshop of Grid Test-bed held in conjunction with ccGrid'06.

[15] Skype, www.skype.com

[16] Putchong Uthayopas, Sugree Phatanapherom, Thara Angskun, Somsak Sriprayoonsakul, "SCE: A Fully Integrated Software Tool for Beowulf Cluster System", Proceedings of Linux Clusters: the HPC Revolution, National Center for Supercomputing Applications, University of Illinois, Urbana, IL, June 25-27, 2001.

[17] P. M. Papadopoulos and M. J. Katz and G. Bruno, "NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters", Proceedings of 2001 IEEE International Conference on Cluster Computing, Newport, CA, Oct-2001.