Evaluating the Information Power Grid using the NAS Grid Benchmarks

Rob F. Van der Wijngaart* and Michael A. Frumkin
NASA Advanced Supercomputing Division
M/S T27A-1, NASA Ames Research Center
Moffett Field, CA 94035-1000
{wijngaar,frumkin}@nas.nasa.gov

NAS Technical Report NAS-04-005

May 2004

Abstract

The NAS Grid Benchmarks (NGB) are a collection of synthetic distributed applications designed to rate the performance and functionality of computational grids. We compare several implementations of the NGB to determine programmability and efficiency of NASA's Information Power Grid (IPG), whose services are mostly based on the Globus Toolkit. We report on the overheads involved in porting existing NGB reference implementations to the IPG. No changes were made to the component tasks of the NGB.

1 Introduction

As computational grids are gaining more acceptance and prominence, tools are required to determine which components of grids function well and which require improvement. To do this in a systematic way, a standard rating mechanism must be developed, i.e. grid benchmarks. Our approach is to develop such benchmarks primarily to serve grid users, since increased application programmer productivity and application performance are the main goals of computational grids. Consequently, we have focused on characterizing actual distributed applications that are suitable for execution on grids. The outcome of that work, the first publicly available grid benchmark suite, was released under the name NAS Grid Benchmarks (NGB), whose precise specification is described in [7]. The motivation, background and early experiences with NGB are reported in [3], along with a brief description of a reference implementation in Java.

In this paper we discuss two implementations of NGB on NASA's production computational grid called the Information Power Grid (IPG), and compare that to the earlier serial implementation. The performance results are reason for guarded optimism that IPG may be beneficial for NASA's application workload, but programmability and reliability can still be improved.

The rest of this paper is structured as follows. In Section 2 we briefly review the NGB, including the non-IPG reference implementations. We also discuss some other grid benchmarking and monitoring projects. In Section 3 we describe the software and hardware infrastructure of IPG relevant to our experiments. In Section 4 we give important details of our actual IPG NGB implementations. We discuss performance and programmability results in Section 6.

Some concluding remarks about the current state of the IPG, as well as recommendations for enhancement, are presented in Section 7.

2 Background

The NGB capture a subset of distributed applications that can be cast in the form of data flow graphs (DFGs). Such applications are among the simplest in terms of software infrastructure required, and hence they should be represented in a basic grid benchmarking suite. The DFGs for these benchmarks, named Embarrassingly Distributed (ED), Helical Chain (HC), Visualization Pipe (VP), and Mixed Bag (MB), are depicted in Figures 1 and 2. The nodes of the graphs, indicated by the rectangular boxes, are computa-

*Employee of Computer Sciences Corporation. This work was supported by the NASA Computing, Information, and Communications Technology (CICT) program, and performed under Task Order A61812D (ITOP Contract DTTS59-99-D-00437/TO #A61812D) awarded to Advanced Management Technology Incorporated.

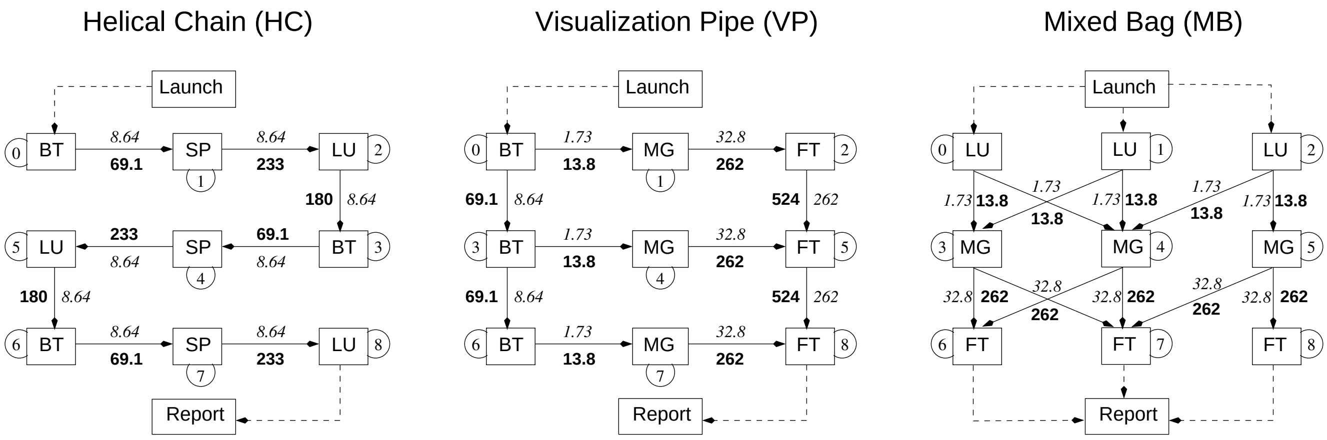

Figure 2. Data flow graphs representing HC, VP, and MB NAS Grid Benchmarks; numbers in *italics* and bold indicate thousands of words communicated between tasks for Classes S and W, respectively. Tasks are numbered according to the labels in the semi-circles.

tional tasks. Dashed arrows indicate control flow between the tasks, and solid arrows indicate data as well as control flow. Communication volumes between tasks are indicated in Figure 2. Launch and Report do little work; they initiate execution of the task graph and collect performance and verification results, respectively. The other computational tasks, SP, BT, LU, MG, and FT, have some internal structure, see Figure 3 in Section 4. They are derived from the NAS Parallel Benchmarks (NPB) and involve computations on sizable multi-dimensional arrays. When the arrays on connected nodes are of the same size, no data transformation needs to take place, but if a node has more than one input arcs and/or if the arrays on connected nodes are of different size, an additional computation needs to occur to merge and/or interpolate data. This computation is carried out by a function called Mesh Filter [7]. The NGB are parameterized and can be run for different array sizes, which are usually referred to as Classes. In this study we use the two smallest Classes, named S and W.

ED, HC, VP, and MB highlight different aspects of computational grids. ED requires virtually no communication, and all the tasks within the graph can be executed independently. It tests basic functionality of grids and does not tax their communication performance; but it does allow us to measure the cost of remote task creation. HC is totally sequential at the graph level. Hence, any time spent on communicating data between graph nodes is fully exposed and will show up in performance results. VP requires specifying concurrent execution of tasks in a DFG with nontrivial dependencies. It allows pipelining and overlapping communication times with computational work. MB is similar to VP, but the nodes of the graph all have different amounts of computational work.

Other projects in the area of grid benchmarking and evaluation have thus far mainly concentrated on monitoring health and status of particular grid components. Examples are the Globus Heart Beat Monitor [10] (part of the Globus Toolkit [11]), the Network Weather Service [8], and Wren [9]. Recently, some efforts were made to determine which realistic applications and application classes qualify as potential grid benchmark examples [6, 12], but no final selections have yet been made. The GRASP project [2] provides a collection of low level probes that mostly measure the network capabilities of grids. It distinguishes itself from most other grid measurement projects in the sense that the probes are formulated and implemented as complete grid applications, including authentication, validation, and resource verification.

3 Information Power Grid

The Information Power Grid [5] was conceived to unify NASA's many and geographically dispersed computational resources. It contains a number of tools and components in its layered architecture, for work flow creation and management, scheduling, etc. At present much of the middleware is provided by the Globus Toolkit, version 2.4.2 [11], and this is the software that we use for implementing NGB on the IPG. In our experiments we also investigate the file transfer properties of two protocols that are not part of GT2.4, but that can be used with it: GridFTP, based on `gsincftp` tools (version 3.0.6), and `gsiscp` version 3.4. GridFTP, `gsiscp`, and Globus all use the same authentication method (Grid Security Infrastructure GSI).

Systems that are currently part of the IPG include four SGI Origin 2000 (O2K) and two Origin 3000 (O3K) systems at NASA Ames in California, a PC Cluster and an O2K system at NASA Glenn in Ohio, and two O2K and one O3K systems at NASA Langley in Virginia [13]. For this study we use just the Origin systems at Ames.

4 Implementations

We describe two Globus implementations of NGB. The first, called NGBIPG-Ksh, uses Korn shell scripts to launch the tasks in the DFGs and to control data transfers. It is a direct extension of the serial NGB reference implementation using a single local file system, available from NAS as part of the GridNPB3.0 package [14]. We will refer to this serial implementation as SHF-Ksh. The second Globus implementation, called NGBIPG-Jav, uses Java code to communicate directly with GRAM [11] and GridFTP clients. Both Globus implementations use Fortran executables to do the computational work. At present these executables are serial themselves, but several can run simultaneously, depending on the parallelism present in the NGB DFGs.

Most of the Origin systems at NAS employ schedulers that act as the default Globus jobmanagers. To prevent jobs from getting stalled in queues we specify the fork jobmanager for all tasks. This is possible because all submitted jobs are of short duration and use minimal machine resources. Hence, they can run in machine partitions usually dedicated to interactive debugging and administrative operations. Once the larger NGB Classes are run, however, individual NGB tasks will require significantly more machine resources, and hence must be queued. Since HC, VP, and MB feature dependencies between tasks, independent scheduling of tasks would be deleterious to IPG performance, and an approach must be taken that submits an entire NGB task graph—including all dependencies—to a grid scheduler, for example as it is done in the Grid Navigation System, [4], or in GRID Superscalar [1].

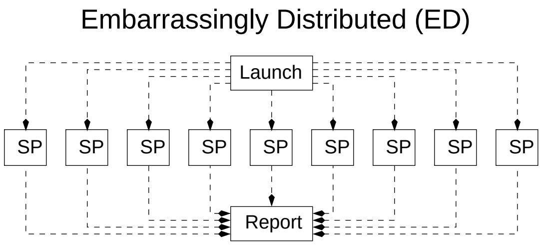

Figure 1. Data flow graph representing ED NAS Grid Benchmark.

4.1 NGBIPG Korn shell implementation

To express the concurrency of ED, VP, and MB we adopt an asynchronous model in which the nodes of the DFGs are issued independently at the NGB launch, using the `globusrun` command with the batch flag "-b." The nodes poll their local file systems for the presence and completeness of the required input files (if any) through the use of semaphores (which are zero byte files themselves). After all nodes are started, the launch script periodically queries their completion using `globus-job-status`. To avoid using too many machine resources, the launch and node scripts poll only once a second.

The structure of the DFG nodes is indicated in Figure 3. Although HC is serial in nature, it can also be implemented

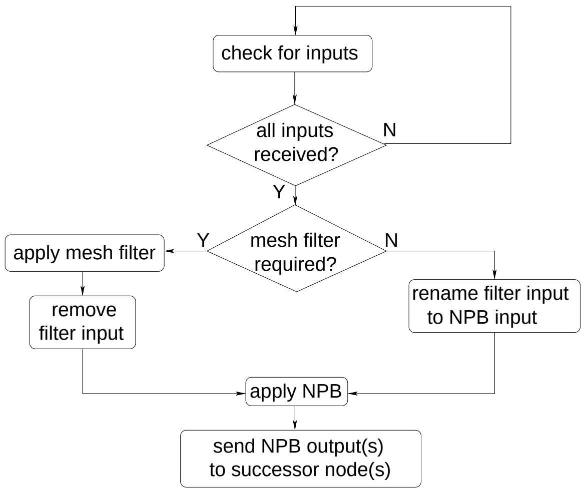

Figure 3. Flow chart of single node of Data Flow Graph.

using the asynchronous model; the nodes must run in sequence, but can be started asynchronously. Hence, we provide two versions of HC, one synchronous, the other asynchronous. In the synchronous version, which is the most natural way of transforming the SHF-Ksh version for execution on the grid, the nodes are launched in blocking mode (i.e. using `globusrun` in non-batch mode). Only when the remote execution request returns is the next node launched. The following sequence describes the actions of the launch process. Run node n on host n; transfer output file from host n to host $n+1$; run node $n+1$ on host $n+1$. Note that this requires third-party copying if host n and host $n+1$ are different from each other and from the host running the launch process. Other than synchronous HC, all NGBIPG-Ksh benchmarks allow the choice between GridFTP and `gsiscp` to transfer files.

4.2 NGBIPG Java implementation

NGBIPG-Jav implementation uses GramJob and GridFTPClient Classes of the cog-1.1 package to access the GRAM and GridFTP servers. The actual access to the SHF-Ksh executables was accomplished by automatic creating shell scripts and coping them to the systems where the executables were invoked. The transfer of files was accomplished using the `transfer` method of the GridFTPClient class and execution of the scripts was accomplished using the `request` method of the GramJob class. Time required for creation of the shell scripts and their transfers to the grid machines was included in the turnaround time of the benchmarks.

For each NGB task we have to copy the following files: the shell script which executes the mesh filter and the actual NPB task, the final report of the task, each of the resulting data files, and the semaphore indicating that the result is ready. We also have to execute the `chmod` command on the remote system to allow execution of the script. Overall, it translates into $2 * (1 + \textit{task output degree})$ accesses to the GridFTP clients and two GramJob requests for each graph node, where *task output degree* equals the number of output files. We estimate that our code adds 1-2% overhead to the actual benchmark turnaround time in the form of time spent to write scripts and to poll internal job submission threads.

The submission of the NGB tasks was performed in a data flow manner. In other words, a task was submitted for execution on a statically specified host as soon as all its predecessors had finished their work and transferred the necessary files to the target machine. For ED it translates into simultaneous submission of all tasks; for HC into synchronous submission described in the previous subsection; and for VP and MB into submission of all tasks that were dynamically determined to be ready.

4.3 Common implementation problems

We identified the following issues in the development of the IPG versions of NGB, independent of the API chosen.

Preliminaries:

- Gridmap files containing IPG user information need to be updated on all grid machines by various system administrators before a proxy can be used there. The procedure is not transparent to the user. Messages from various administrators are confusing.

- Globus jobs can only be started from systems on which the user has obtained a proxy. This makes access to the grid asymmetrical, and the user has to remember where the proxy was obtained.

- It is possible to obtain a proxy on a host by copying the gridmap file from a machine on which an earlier proxy was obtained. This constitutes a security breach.

File transfer:

- GridFTP
  1. Does not allow third-party copying of files.
  2. Does not allow the renaming of the target file in the `gsincftp` command line interface (but does allow it in the Java API).
  3. Does not retain the access permissions on files at the destination. For executables this necessitates issuing a "`chmod`" command after the file transfer completes, with the concomitant overhead of a remote execution.
  4. Will clobber a file that is copied onto itself, while no mechanism to avoid clobbering is provided.

  Elimination of the first and second GridFTP restrictions is highly desirable for flexible programming. Third-party copying can always be accomplished through two two-party copies, but this is costly and usually leads to sequential bottlenecks. Target file renaming is useful, for example, in the semaphore technique used in NGBIPG-Ksh to signal successor nodes in the DFG that inputs have arrived. If a semaphore of the same name must exist on the sending host, it will not be possible to determine whether sending and receiving host share the same file system (as is the case for a number of hosts at NAS). Hence, a file transfer that is intended to be a move (destroy the copy at the source if sender and receiver use different file systems) cannot be safely implemented. Item 4 is especially harmful, as for the user it is often difficult or costly to determine whether two files on different hosts are actually the same. Safety measures to avoid accidental destruction of files are cumbersome to implement.

- `globus_URL_copy`
  1. Suffers from most of the ills that GridFTP does.
  2. Requires knowledge of absolute paths of files on the various hosts on which grid users have accounts, since there is no concept of user home directories. This severely inhibits transparency of grid use and makes applications non-portable.

- `gsiscp`
  1. This file transfer protocol has all the properties desirable for convenient programming (target file renaming, no clobbering, persistence of permissions, third-party copying), but is reported to be

slower than GridFTP and globus_URL_copy (however, see Section 6 for a comparison).

Execution environment:

- The exit status of a remotely executing command is not available to the globusrun command that launched it. It has to be stored in a file and transferred to the place from which the globusrun command was issued.
- If the fork jobmanager is used to launch a remote script, the user's shell resources files are not sourced, which means that no path will be available, except that to globus commands through the $GLOBUS_LOCATION environment variable. If the batch scheduler jobmanager (Portable Batch System at NAS) is selected, resource files will get sourced. This means that jobs may succeed or fail, depending on what jobmanager is selected. A solution is always to source resource files in scripts executing through globusrun.

5 RMI Implementation

In this section we describe an implementation of the NGB using Java Remote Method Invocation (RMI) mechanism. This mechanism we used in Java implementation of the NAS Grid Benchmarks released in 2002. In addition we used Java Native Interface (JNI) to run native executables and newly developed FileContent class for reading, transferring and writing binary files. These additions did not exceed 1000 lines of the code. Most of the code was used for detection of location executables and temporary files on the hosts. In the implementation we used same GridNPB3.0 executables as we used in the IPG implementations. The execution results are shown in the Table 2.

Table 2. Turnaround times (sec) for RMI implementations of NGB.

Class	configuration	ED	HC	VP	MB
S	*boyd*	4.3	11.9	43.7	17.2
	dean	2.8	8.7	24.3	10.4
	duo	2.4	5.2	34.4	11.3
	mix	1.5	7.0	40.4	15.5
W	*boyd*	506.7	170.9	114.7	98.2
	dean	48.7	77.6	62.5	45.5
	duo	275.3	125.0	85.5	79.8
	mix	142.7	135.3	106.3	62.3

For user to be able to execute a benchmark task of a host we have adopted BenchmarkServer of the NAS Grid Benchmarks. The security policy for the server and server itself has to be installed by a grid administrator. The server binds to the RMI registry on a host and listens for request to execute a benchmark task on it. If a request for task execution arrives (as a result of calling a method of a remotly obtained handle to the BenchmarkServer) the server executes the following methods :

- for each incoming arc it waits data to arrive
- it executes the Mesh Filter on the arrived arrays to produce initial values for the benchmarking task
- it executes the startBenchmark method of the task to perform the benchmark
- it reads the results of the task
- it sends the results and output arrays along all outcoming arcs of the task

The interface with with the native executables implementing the Mesh Filter and an NPB task is accomplished by using the JNI mechanism which allows to pass parameters to a native function and execute it. The native interface is the only function which has a machine dependent compilation (into a shared library). All other code is compiled in Java 1.1 class format on any machine and can be installed all hosts using the class files. Since the location of the executables and of the scratch directory used for intermediate files vary from host to host each benchmark server has a mapping information to map the host names to the locations. The field files generated by the executables are read byte by byte by the read method of a FileContent object and the content is transferred by the BenchmarkServer to another host via the RMI mechanism. To preserve correct endian the bytes of each word are swapped when transferred between machines with opposite endians.

6 Results

We present performance results for the following experiments and configurations. To ensure repeatability no grid schedulers were used; we explicitly prescribe the machines on which the tasks are to be run. The machines used in our experiments are listed in Table 1.

Configurations *boyd* and *dean* consist of just those machines, respectively. Configuration *duo* consists of dean and boyd. Configuration *mix* consists of dean, boyd, alan, dean, joe, grace, dean, harv, and boyd. Tasks in the DFGs are mapped round robin to the sequence of machines in these configurations.

The turnaround time does not include deployment (copying) of executables; it does include deployment of scripts and the verification.

Table 1. Grid Machines. The file systems of boyd, alan, joe, grace, and harv are cross-mounted, while dean has its own file system.

Name	Clock Rate (MHz)	Architecture	Number of Processors	Processor Type
boyd	250	O2K	16	MIPS R10000
alan	250	O2K	32	MIPS R10000
grace	250	O2K	128	MIPS R10000
joe	400	O2K	128	MIPS R12000
harv	400	O3K	512	MIPS R12000
dean	600	O3K	1024	MIPS R14000

Table 3. Turnaround times (sec) for IPG (NGBIPG) and non-IPG (SHF) implementations using Korn shell scripts. Here *conf* indicates configuration, `loc` refers to running all tasks locally from a single script (SHF-Ksh). HC `gsiscp`-sync refers to the explicitly synchronous version. MB: see footnote on p.6.

		ED		HC				VP		MB	
Class	*conf*	GridFTP	`gsiscp`	GridFTP	`gsiscp`	`gsiscp` sync	`loc`	GridFTP	`gsiscp`	GridFTP	`gsiscp`
S	*boyd*	38	40	33	33	55	2	34	34	38	36
	dean	32	30	27	27	47	1	28	27	31	31
	duo	36	34	51	29	78		50	31	33	34
	mix	39	37		37	115			51	37	36
W	*boyd*	656	650	159	159	214	128	84	83	87	86
	dean	71	69	59	58	129	50	39	39	41	42
	duo	273	273	142	112	203		81	62	65	65
	mix	171	172		157	283			88	56	56

6.1 Korn Shell

In the experiments reported in Table 3 we always used `gsiscp` to copy semaphore files between hosts (GridFTP was not satisfactory, since its command line interface does not allow file renaming). An alternative to copying is to create semaphore files directly on remote systems by issuing `touch` commands through `globusrun`. However, in experiments with HC on configuration *duo*, the `touch` semaphore approach generally showed significantly poorer performance than `gsiscp`, as well as much greater variability. For instance, 28 interleaved runs of HC Class S with both approaches showed average turnaround times that were 3.9 seconds longer for `touch` than for `gsiscp`. And whereas variability with `gsiscp` was within 17%, `touch` produced differences of up to 57%.

In our first NGBIPG-Ksh implementation, which did not yet include MB, we did not provide methods to determine which hosts' file systems were cross-mounted and which were not. Consequently, some redundant copying (overwriting) of files occurred in the `gsiscp` HC and VP experiments on configuration *mix*. We did not obtain performance results for the GridFTP versions of VP and HC on configuration *mix*, since file clobbering occurred on the cross-mounted file systems[1].

As expected, in the single-system configuration experiments, *dean* performed significantly better than *boyd*, since dean has much faster processors. Moreover, ED, which has the highest level of concurrency, quickly saturates the CPUs available to the fork jobmanager on boyd, whereas dean, which has many more processors, can accommodate all nine tasks simultaneously without performance deterioration. The reason why ED Class S does not benefit as much from the faster processors as does Class W is that its performance is dominated by remote process creation and polling for results. This is demonstrated by comparing the IPG and non-IPG (`loc`) performance results of HC Class S; virtually all execution time is spent on process manage-

[1] Once we had implemented MB and supplied it with a mechanism to detect file systems shared by hosts, the IPG had changed and *all* configurations shared the same file system. Thus, in the *duo* and *mix* configurations of MB no redundant copying of files ever occurred, and accidental clobbering of files could be avoided. We did not rerun all other benchmarks using the same technique, because then we would not have been able to test the file transfer properties of `gsiscp` and GridFTP.

ment in the IPG implementation. The effect of saturation is highlighted by the ED Class W experiments on configuration *duo*. Even though performance of ED is determined, in principle, by the slowest system in the configuration (*duo* utilizes both boyd and dean), turnaround time on *duo* is less than half that on *boyd*.

While HC is synchronous in nature, the asynchronous implementation always runs fastest. The reason is that remote process creation can be overlapped with computation, which the synchronous version does not allow.

The combined results of the HC and VP runs show that `gsiscp` is significantly faster than GridFTP on our testbed. When files are shared by tasks on the same host, we never copy, which explains why the `gsiscp` and GridFTP results on configurations *boyd* and *dean* are not affected by the file transfer protocol. However, on configuration *duo*, where two hosts with different file systems are used, VP with `gsiscp` is 19 seconds faster than with GridFTP for both benchmark classes. For HC, which cannot hide transmission costs, the difference is even larger. Deployment of executables and scripts is not reported in Table 3, since it takes place before the actual job launch. However, we did measure these file copy speeds, and for HC and VP, GridFTP was between four and six times slower than `gsiscp`. For ED the difference was a factor of three to four.

6.2 Java

Turnaround times of the NGBIPG-Jav implementation are presented in Table 4. As was mentioned in Section 4, we perform $2*(1+\textit{task output degree})$ accesses to the GridFTP clients and make two GramJob requests within each graph node (Because of the shared file system on some machines, not all GridFTP calls actually result in a file copy.) For Class W of HC and VP we estimate that about 2/3 of the turnaround time is spent accessing and delivering these IPG services, and 1/3 of the time for the actual processing. Access and use of the services mostly involves network and memory subsystems, which blunts the raw processor speed advantage of dean. Otherwise, the same arguments used to explain the NGBIPG-Ksh turnaround times are valid.

We have to be careful when interpreting the performance results, because a number of factors were beyond our control and not all could be measured independently. These include actual machine load, network traffic stability, and interference with other IPG users, setup of the IPG services (timeout, polling intervals etc.). To reduce influence of these random factors we repeat our experiments multiple times and report the best results (shortest turnaround times). Variation of turnaround times among runs reported in Tables 3 and 4 was within 20%. Also, a number of our experiments failed because of IPG authentication failure. In these cases we just repeat the experiment with a 10 minutes delay.

Table 4. Turnaround times (sec) for IPG (NGBIPG) implementations using Java. Here *conf* indicates configuration.

Class	*conf*	ED	HC	VP	MB
S	*boyd*	104	251	205	165
	dean	83	242	176	117
	duo	101	255	192	139
	mix	93	271	200	183
W	*boyd*	660	320	243	228
	dean	132	247	179	165
	duo	404	296	231	203
	mix	230	346	214	204

Volatility of the performance of the grid machines is the major factor which can distort the performance of the IPG benchmarks. To gauge performance volatility we used the non-IPG Java version of the NGB [14] and monitored grid machines during some of our experiments. The monitoring results using ED.S, depicted in Figure 4, show that performance of the grid machines usually is within 20% of the average when all machines are up and the benchmark servers are properly installed.

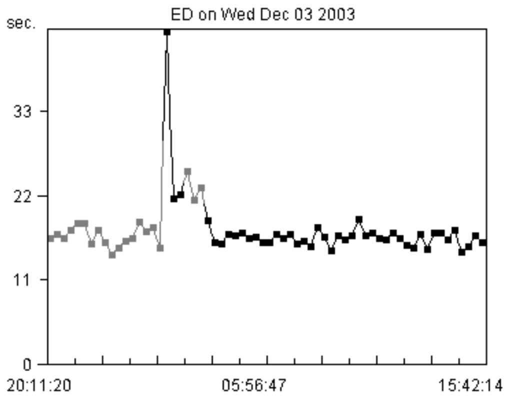

Figure 4. Typical turnaround time of Java ED.S benchmark running on nine grid machines. Grey dots indicate benchmark did not pass verification because of machine unavailability. Large peak indicates a setup process of a fresh server on one of the grid machines.

7 Conclusions

It is important to recognize that our performance experiments are not exhaustive, and are focused on determining overheads of currently implemented IPG services, which are likely to improve over time. The experiments, however, demonstrate the use of NGB to measure grid characteristics and as a pathfinding tool for grid application developers.

One important aspect, programmability of the IPG, was found to be in need of some improvement. Unexpected file clobbering by GridFTP, absence of third-party copying, and lack of functionality of its command line interface (no file renaming) leads to cumbersome programming. The exit status of jobs executed through `globusrun` cannot be communicated to the calling process, other than through explicit capture and file transfer by the programmer. `globus_url_copy` requires explicit knowledge of remote absolute paths, which hampers portability. Execution of Unix commands through `globusrun` requires knowledge of the paths to these commands on remote machines, which also hampers portability.

References

[1] R.M. Badia, J. Labrata, R. Sirvent, J.M. Pérez, J.M. Cela, and R. Grima. *Programming Grid Applications with Grid Superscalar.* http://www.zib.de/ggf/apps/meetings/ggf8/badia1.pdf

[2] G. Chun, H. Dail, H. Casanova, A. Snavely. *Benchmark Probes for Grid Assessment.* UCSD Technical Report CS2003-0760, University of California, San Diego, CA, 2003.

[3] M.A. Frumkin, R.F. Van der Wijngaart. *NAS Grid Benchmarks: A Tool for Grid Space Exploration.* Cluster Computing Vol. 5, No. 3, pp.- 247-255, 2002.

[4] M.A. Frumkin, R. Hood. *Using Grid Benchmarks for Dynamic Scheduling of Grid Applications.* Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'2003), Marina Del Rey, USA, Nov. 3-5, 2003, p. 31-36, NAS Technical report NAS-03-015.

[5] W.E. Johnston, D. Gannon, B. Nitzberg. *Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid.* Proc. 8^{th} IEEE Intl. Symp. High Performance Distributed Computing, Redondo Beach, CA, August 1999.

[6] A. Snavely, G. Chun, H. Casanova, R. Van der Wijngaart, M.A. Frumkin. *Benchmarks for Grid computing: A Review of Ongoing Efforts and Future Directions.* SIGMETRICS Performance Evaluation Review (PER), Vol. 30, No. 4, March 2003.

[7] R.F. Van der Wijngaart, M.A. Frumkin. *NAS Grid Benchmarks Version 1.0.* NAS Technical Report NAS-02-005, NASA Ames Research Center, Moffett Field, CA, 2002.

[8] R. Wolski, N.T. Spring, J. Hayes. *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.* J. Future Generation Computing Systems, 1999.

[9] M. Zangrilli, B.B. Lowekamp. *Comparing Passive Network Monitoring of Grid Application Traffic with Active Probes.* SIGMETRICS Performance Evaluation Review (PER), Vol. 30, No. 4, March 2003.

[10] *Globus Heartbeat Monitor.* www.globus.org/hbm/.

[11] *Globus Toolkit.* www.globus.org.

[12] *GRIDBench.* www2.cs.ucy.ac.cy/~georget/gridb/gridb.html.

[13] *NASA's Information Power Grid.* www.ipg.nasa.gov.

[14] *NAS Grid benchmarks.* www.nas.nasa.gov/Software/NPB.