Measuring the Performance and Reliability of Production Computational Grids

Omid Khalili [1], Jiahua He [1], Catherine Olschanowsky [2], Allan Snavely [2], Henri Casanova [3]

[1]*Dept. of Computer Science and Engineering, University of California, San Diego*
9500 Gilman Dr
La Jolla, CA 92093-0505, USA
{okhalili,j2he}@cs.ucsd.edu

[2]*San Diego Supercomputer Center*
9500 Gilman Dr
La Jolla, CA 92093-0505, USA
{cmills,allans}@sdsc.edu

[3]*Dept. of Information and Computer Sciences, University of Hawai'i at Manoa*
1680 East-West, Rd
Honolulu, HI 96822, USA
henric@hawaii.edu

***Abstract*— In this work we report on data gathered via a deployment of a monitoring and benchmarking infrastructure on two production grid platforms, TeraGrid and Geon. Our result show that these production grids are rather unavailable, with success rates for benchmark and application runs between 55% and 80%. We also found that performance fluctuation was in the 50% range, expectedly mostly due to batch schedulers.**

We also investigate whether the execution time of a typical grid application can be predicated based on previous runs of simple benchmarks. Perhaps surprisingly, we find that application execution time can be predicted with a relative error as low as 9%.

I. Introduction

Computational grids subsume traditional compute, storage, and data acquisition resources by federating them. These platforms hold the promises of increased capacity and performance, which users/applications should achieve by the use of high-level software abstractions. However, under the covers many resources and middleware services are involved and although the high-level interface may be convenient it is difficult to understand the performance and availability characteristics of such systems without appropriate benchmarking. There is no one performance or reliability of a grid; there is only a continually evolving time-series of performances and reliabilities that may be observed and recorded.

A broad question is then "How do performance and reliability vary in production grids?" and we make four contributions: (i) we present a generic grid measurement infrastructure, which we have deployed on two state-of-the-art grids for several months; (ii) we quantify the availability of the hardware and the middleware infrastructure in both platforms; (iii) we quantify the magnitude and the sources of performance fluctuations in both platforms; and (iv) we find that the performance experienced by simple "benchmark probes" can be used to predict the performance of a typical application with relative error as low as 9%.

We used Inca [1], a test harness for periodically capturing and recording time-series of grid performance and availability metrics. Using Inca we deployed the GrASP (Grid Assessment Probes) [2] benchmark probes and the PreCo (from [3]) application on two state-of-the-art grid platforms: TeraGrid [4] and Geon [5]. Sections II, III and IV describe our measurement infrastructure, target platforms, and experimental methodology. Section V presents detailed performance and reliability data for both grids. Section VI presents and evaluates application performance prediction methods. Section VII discusses related work and Section VIII concludes the paper with a brief summary of results.

II. Measurement Infrastructure

We found that, in order to collect the type of data needed to characterize the performance and reliability of a computational grid in a meaningful way three components are required: (i) a set of benchmark probes that exercise basic grid functionality and that collect timing information and error messages as they run; (ii) an actual Grid application for comparison to the probes; and (iii) a framework for periodically running the probes and application, for archiving results, and for providing a way to query the results. These requirements were met by the GrASP probes, PreCo, and Inca respectively.

A. Grid Assessment Probes

The Grid Assessment Probes (GrASP) [2] are designed to serve as simple grid application kernel exemplars as well as a set of diagnostic tools. They test and measure performance of basic grid functions including file transfers, remote execution, and Grid Information Services response. All probes perform the same set of initial operations (check for a valid grid

proxy, authenticate to all involved resources, check for disk space availability, etc.). In this paper we use the two probes described below

Circle Probe – The Circle probe takes a 100 MB file and passes it in a ring around a given set of grid nodes, performing a checksum at each step along the way to ensure that the file has come across intact. As a final step, the file is transferred back to the originator, and a simple diff is applied to validate that the file is identical to the original. There can be any number of nodes involved in the probe. This probe is meant to emulate an application performing a token-passing operation around grid sites.

Gather Probe – The Gather probe transfers 100 MB data files in parallel from any number of source nodes to a single compute node. A computation is performed on the input files and a single 500MB output file is generated and transferred to a single destination site. This probe is meant to emulate an application that performs a data aggregation operation across grid sites.

B. PreCo

PreCo (also called Transformed based Base Projection (TxBR)) is a research code used by the National Center for Microscopy and Imaging Research [3]. The computation component of this code uses a back projection algorithm to take 2-D images collected from an electron microscope to generate 3-D images. PreCo is a good exemplar of grid applications because data acquisition is potentially geographically separated from computation, computation is essentially embarrassingly parallel (thus able to efficiently utilize distributed parallel computing resources), while visualization from the results of computation is again potentially geographically separated from the previous two steps.

C. Inca

Inca [1] is a flexible framework for the automated testing, benchmarking and monitoring of Grid systems. Originally developed for use within the TeraGrid [4] project Inca has been generalized and is in use on other computational grids including Geon [5] and DEISA [6], it is included with the NMI [7] R7 release. Inca includes mechanisms to schedule the execution of information gathering scripts, and to collect, archive, publish, and display data. Inca supports a diverse set of use cases including, service reliability verification, monitoring, benchmarking, site interoperability certification, and software stack validation.

For gathering data Inca makes use of a Reporter functionality further described in [8]. A Reporter interacts directly with a resource to perform a test, benchmark, or query. For example, a Reporter can publish the version of a software package or perform a unit test to evaluate software functionality. We obtained an alpha version of Inca 2.0 prior to its official release and used a configuration that included a single Reporter Manager running on tg-login.sdsc.teragrid.org, three Reporters (which we implemented for the two GrASP probes and for PreCo), and the Depot which is the Inca measurement database.

III. Testbeds

TeraGrid – TeraGrid [4] aggregates resources at eight partner sites to create an integrated, persistent computational grid. Deployment of TeraGrid was completed in September 2004, bringing over 40 teraflops of computing power and nearly 2 petabytes of rotating storage into production, interconnected at 10-30 gigabits/second via a dedicated national network. All resources run the TeraGrid Common Software Stack (CTSS) which includes the Globus Toolkit version 2 (GT2). The resources in the grid include mostly clusters of Itanium2 IA-64, Alpha EV68, Itanium IA-32, IBM Power3-II.

Geon – The GEON grid [5], targeted to Earch Sciences applications, aggregates resources over fifteen instutions. Each institution runs a "GEONgrid" host, which provides an entry point into the system, runs a reference GEON software stack which includes the Globus Toolkit version 3 (GT3), and may be a gateway to local Data nodes and/or Compute nodes. The hardware consists of commodity X86 systems purchased from Dell and ProMicro with various racks and switches from Dell.

IV. Experiments

Data was collected on both TeraGrid and Geon resources between September 6, 2005 and March 20, 2006, providing us with approximately 6 months of data on the TeraGrid and 3 months of data on GEON.

A. Inca Configuration

TABLE I
Resources used in GrASP and PreCo Configurations on TeraGrid.

Hostname	Physical Location
tg-grid1.uc.teragrid.org	ANL, Chicago, IL
tg-login1.iu.teragrid.org	IU, Bloomington, IN
tg-login.ornl.teragrid.org	ORNL, Oak Ridge, TN
lonestar.tacc.utexas.edu	TACC, Austin, TX
tg-login.purdue.teragrid.org	Purdue, West Lafayette, IN
tg-login.ncsa.teragrid.org	NCSA, Urbana-Champaign, IL
tg-login.sdsc.teragrid.org	SDSC, San Diego, CA

1) TeraGrid Resources: The Gather probe was run over the seven TeraGrid Resources shown in Table I. Data source nodes were ANL, IU, ORNL, TACC, and Purdue, the compute node was NCSA and the destination node was SDSC. The NCSA compute node has a cluster of Intel(R) Itanium2(R) 1.3 GHz processors. A 100 MB file was generated and transfered from each of the source nodes to NCSA where a computation was executed and a file approximately 500 MB in size was then transfered to SDSC. The Circle probe included the same seven nodes, each transferring a 100 MB file to the next. The order of transfers was IU, ORNL, Purdue, TACC, SDSC, NCSA, ANL and then back to IU.

The PreCo application was run on TeraGrid across a smaller set of resources including Purdue, NCSA and SDSC. Data was sent from Purdue to NCSA where a computation took place and the result was then sent to SDSC for storage.

TABLE II
RESOURCES USED IN GRASP AND PRECO CONFIGURATIONS ON GEON.

Hostname	Physical Location
utepgeon01.utep.edu	UTEP, El Paso, TX
agassiz.la.asu.edu	ASU, Phoenix, AZ
geon06.sdsc.edu	SDSC, San Diego, CA
geonnet1.mines.uidaho.edu	UI, Moscow, ID
geongrid.rice.edu	RICE, Houston, TX
geongrid.geo.arizona.edu	Arizona, Tucson, AZ
cgrid0.geol.iastate.edu	ISU, Ames, Iowa

2) Geon Resources: We ported the GrASP probes to use GT3. The Web Services (WS) GRAM server is used to submit jobs to remote resources and GridFTP is used to transfer files (as on the TeraGrid). At the time that we started the benchmarks, the PBS scheduler on the compute resource was not configured; instead all jobs were forked and ran on the login node. PreCo requires its compute processes to be submitted to a batch scheduler, and thus we were unable to schedule PreCo on Geon. The PBS scheduler has recently been configured and will be used in future tests.

Both the Gather and Circle probes were run on seven Geon resources. For the Gather probe, we used UTEP, ASU, SDSC, UIDAHO and RICE as data sources. The compute node is Arizona and the results node is CGRID. The login node of the compute node, Arizona, has two hyperthreaded Intel(R) Xeon(TM) 2.80GHz processors. The same source file and results file sizes are used as on the TeraGrid. The Circle probe has the same transfer file size as on TeraGrid and has the following transfer order: ASU, SDSC, UIDAHO, RICE, Arizona, CGRID, UTEP and finally back to ASU.

B. Schedule Configuration

Initially the probes were scheduled to run hourly one after the other. The application was run simultaneously with the probes, immediately following the probes and thirty minutes after the probes in order to find any instantaneous, short delay, or long delay correlations between runtimes. This schedule was repeated continuously throughout the data gathering period. After approximately three months of measurements were taken with this schedule, we changed it to run Gather and PreCo simultaneously every half hour in order to collect more data for evaluating application performance prediction methods (see Section 6).

V. PERFORMANCE AND AVAILABILITY

A. TeraGrid

1) Performance: Measuring the performance of the GrASP probes on TeraGrid involved measuring the time spent transferring 500 MB of data to a compute site (100 MB from five separate sites in parallel), time spent in batch scheduler queues, time spent computing and time spent transferring a 500 MB result file.

Tables III and IV show statistics for each step of the Gather probe over six months. Table III shows statistics for GrASP-specific steps, while Table IV shows statistics for those steps that any application would require. The GrASP-specific steps are ones that a real grid application would probably not require on every run (e.g., data generation and staging, compilation). Trends were similar for the Circle probe and are not shown here. We can see in the tables that the two leading causes of variability (i.e., the highest standard deviations relative to the means) are: (i) initialization and finalization operations on the "login" node (initialization, staging data, building executable, and cleanup); and (ii) queue wait times. This was expected as the number of users actively working on a login node varies greatly over time and queue wait times are known to exhibit variable and non-stationary behaviors [9].

TABLE III
STATISTICS ON THE EXECUTION OF GRASP GATHER ON TERAGRID: GRASP-SPECIFIC STEPS ONLY (IN SECONDS).

Statistic	Init.	Stage Data	Build Executable	Cleanup
Average	36.03	38.41	15.35	10.76
Min	26.11	33.48	4.44	3.14
Max	847.85	418.47	70.46	413.46
Stdev	37.14	21.18	3.91	20.37

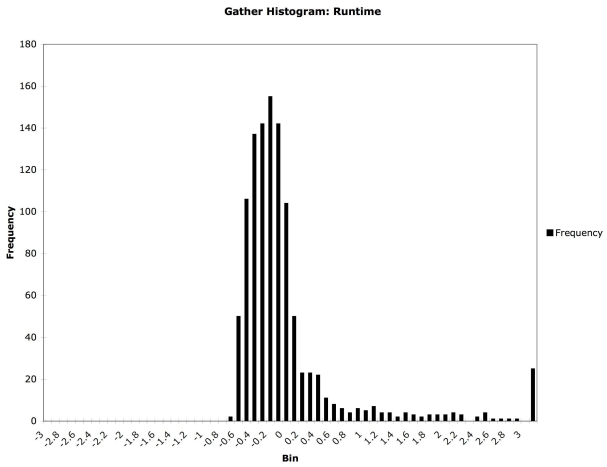

Fig. 1. Frequency of Gather probe runtime on TeraGrid falling between x standard deviations of the mean.

As seen in Table IV we show three separate totals: (i) total probe runtime; (ii) probe runtime excluding setup/cleanup; and (iii) probe runtime excluding setup/cleanup and queue wait times. Total probe runtime has an average of 227.62 seconds and a standard deviation of 116.02 seconds, which is approximately 50% of the average runtime, a large standard deviation. For illustration purposes, Figure 1 shows the frequency of the probe's runtime falling between *x* standard deviations of the

TABLE IV

STATISTICS ON THE EXECUTION OF GRASP GATHER ON TERAGRID: STEPS RELEVANT TO A GRID APPLICATION ONLY AND TOTALS (IN SECONDS).

Statistic	Transfer Data to NCSA	Compute	Queue Wait Time	Transfer Results	Total(i) Runtime	Runtime w/o (ii) Setup/Cleanup	Runtime w/o (iii) Setup/Cleanup/Q
Average	8.38	18.40	84.53	15.72	227.62	127.05	42.51
Min	5.28	17.99	25.94	13.38	145.37	63.51	36.87
Max	194.09	61.21	1008.60	130.53	1142.66	1055.52	260.31
Stdev	9.20	2.04	101.13	7.76	116.02	102.46	14.36

mean. This histogram is skewed to the right; runtimes are clustered near the mean but there is a significant probability of the occasional runtime exceeding the mean by several standard deviations. After removing the setup/cleanup steps, which are specific to the GrASP probes, the total average runtime is 127.05 seconds with a standard deviation of 102.46 seconds. This is 80% of the runtime, an even larger standard deviation due to the fact that queue time is then a larger percentage of the total runtime. As seen in Table IV, when the GrASP-specific stages and the queue waiting times are removed, the total execution time is 42.51 seconds with a standard deviation of only 14.36 seconds or 33% of the mean. This demonstrates that the middleware and network infrastructure of the TeraGrid perform rather consistently; a major factor in observed performance variability comes from queue wait times.

2) Failures: An environment like TeraGrid presents many opportunities for a distributed application to fail. The GrASP probes were configured to run over seven distinct resources in the grid and a successful run required all seven resources to be up and the network connecting them to be operational. We have filtered out all error messages related to scheduled preventive maintenance. The news.teragrid.org web site was used to obtain the dates to be excluded from our measurements.

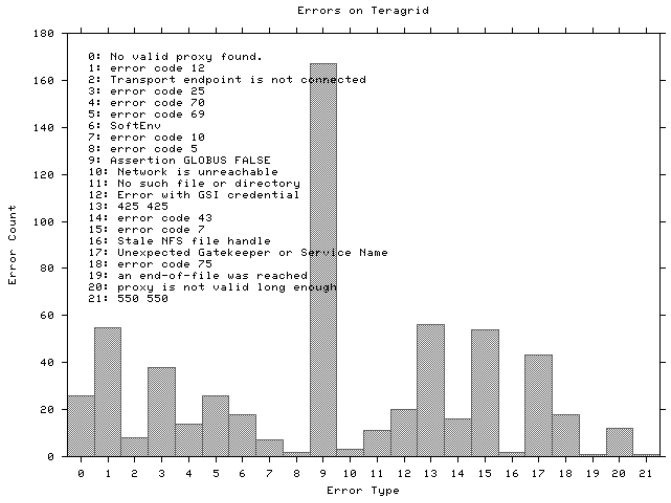

Fig. 2. Number of Gather and Circle Errors on TeraGrid.

Over the first 6 weeks that the GrASP probes where running on TeraGrid resources we observed a success rate of approximately 58% on the gather probe and 78% on the Circle probe. Figure 2 shows the types of errors and their counts. (The error descriptions are cryptic, please see www.sdsc.edu/pmac for detailed error descriptions). Two errors occurred far more often than the rest. The first is a proxy error, which occurred over a period of a few days when the proxy necessary for running the tests was not renewed. This error is considered a user error and we filtered it out when computing overall success rate. The second is a known GT2 error that occurs frequently. The error has been fixed in later releases of Globus (however there are no plans to patch the problem in the version currently in use on TeraGrid).

We found that middleware errors tend to surface for a block of time and after being corrected do not reappear again. Most errors follow this pattern. Only the known GT2 error, softenv error and the error with code 12 occur periodically.

B. Geon

Results on the Geon grid differed qualitatively and quantitatively from those on TeraGrid. The heterogeneity of the TeraGrid presents great challenges to stability and software capability. The Geon grid benefits from dedicated hardware and a homogeneous software stack. However, our deployment uses GT3, which is a recently implemented web services based implementation of GT. The errors and the variability in performance due to this new middleware infrastructure are apparent in our measurements. Furthermore, while TeraGrid has dedicated network links between most of their sites, Geon uses a shared network. Therefore, we see more variability in the transfer times on Geon. Finally, a major difference between the two grids is that Geon does not use batch schedulers but instead timeslices compute resources in an interactive fashion.

TABLE VI

STATISTICS ON THE EXECUTION OF GRASP GATHER ON GEON: GRASP-SPECIFIC STEPS ONLY (IN SECONDS).

Statistic	Init.	Stage Data	Build Executable	Cleanup
Average	96.66	70.26	33.28	74.84
Min	73.30	63.14	29.54	50.14
Max	181.38	139.87	62.04	427.74
STDEV	9.30	8.39	7.50	20.21

1) Performance: Tables V and VI show statistics for the execution of Gather on Geon as for the TeraGrid, minus the time to start the job. For illustration purposes, Figure 3 shows the frequencies of individual execution times for the Gather probe on Geon that fall within x standard deviations of the mean. The statistics displayed in Table VI demonstrate that the time needed for steps on the login node (initialization, staging,

TABLE V

STATISTICS ON THE EXECUTION OF GRASP GATHER ON GEON: STEPS RELEVANT TO A GRID APPLICATION ONLY AND TOTALS (IN SECONDS).

Statistic	Transfer Data to UA	Compute	Job Startup	Transfer Results	Total Runtime	Runtime w/o Setup/Cleanup
Average	29.12	10.58	38.40	15.88	369.01	93.95
Min	21.02	10.15	29.85	14.31	324.80	84.66
Max	81.30	22.67	69.58	189.03	720.24	286.01
STDEV	11.68	0.94	2.14	9.00	34.79	15.63

building the executable and cleanup) are fairly consistent. Compared to the results on the TeraGrid, execution times are longer on Geon but have a lower standard deviation. For example, the initialization step on the TeraGrid, has a average runtime of 36.03 seconds with a standard deviation of 37.14 seconds, whereas on Geon the average runtime is 96.66 seconds with a standard deviation of 9.30 seconds (less than 10%). The runtime on the TeraGrid login nodes for these steps vary so much because they can get heavily loaded, causing occasional longer runtimes, but the average runtime is faster because GT2 is used for job submission. On Geon, the runtime on the login nodes does not vary much because there are only a few users on this new grid; nevertheless the runtime is longer because of the added overhead of using the Web Services GRAM server to start a remote job. Figure 3 and Table V show that a large number of executions fall outside of a standard deviation of the average runtime. This variability is due to the fact that Geon does not have a dedicated network and the runtime of Gather depends on the current network traffic.

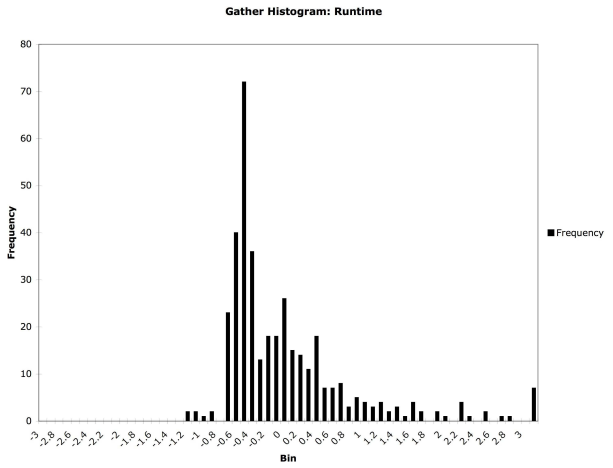

Fig. 3. Frequency of Gather probe runtime on GEON falling between x standard deviations of the mean.

2) Failures: We found that reliability on the Geon grid was much more dependent on the reliability of the middleware than that of the hardware. As mentioned previously we used GT3 which is essentially a web services implementation of GT2. We found a smaller number of error messages being returned from this version of GT. Figure 4 shows the error messages that occurred. The vast majority of errors were difficult to track (either just timing out or not returning any error message). Careful sleuth work uncovered many of these errors are caused by a problem with GT3 cleaning up processes after a failure: once a failure has occurred and GT3 runs out of processes, the software just waits for a connection that cannot be created.

The Gather and Circle probes experienced a combined success rate of 56% on Geon. In most cases once an error with GT3 occurred it took human intervention to correct it and therefore the same error would happen hourly for a long period of time. We found that most errors happened continuously for a few days before being corrected. (www.sdsc.edu/pmac lists the full error messages.) These errors did not occur frequently. As can be seen in the histogram most of them only occur a few times. The errors that occurred many times over large periods of time generally did not generate an error message and they either died silently or the Inca framework timed them out.

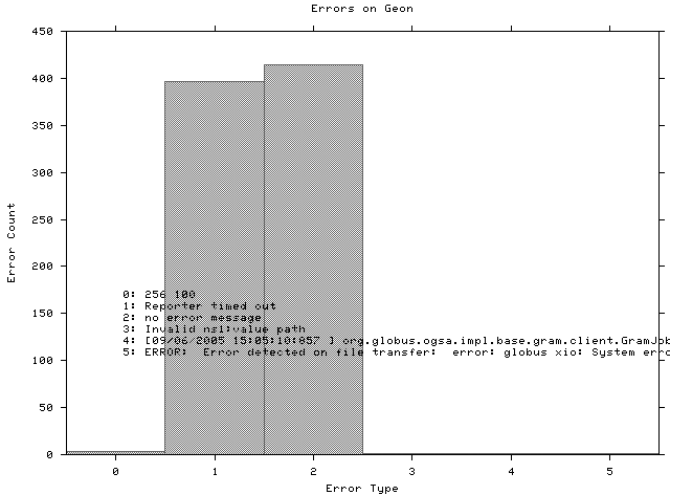

Fig. 4. Number of Gather Errors on Geon.

VI. PREDICTIONS OF APPLICATION PERFORMANCE BASED ON BENCHMARK MEASUREMENTS

The schedule of probe and application benchmark execution included times when the Gather probe and PreCo application were started simultaneously, allowing us to compute the correlation of their performance. We found this correlation, ignoring the GrASP specific steps, to be 0.32. This correlation is low because the queue wait time, which we have shown is a significant source of variability, accounts for most of the overall runtime. The runtimes without the GrASP specific steps and the queue wait time have a correlation of 0.54.

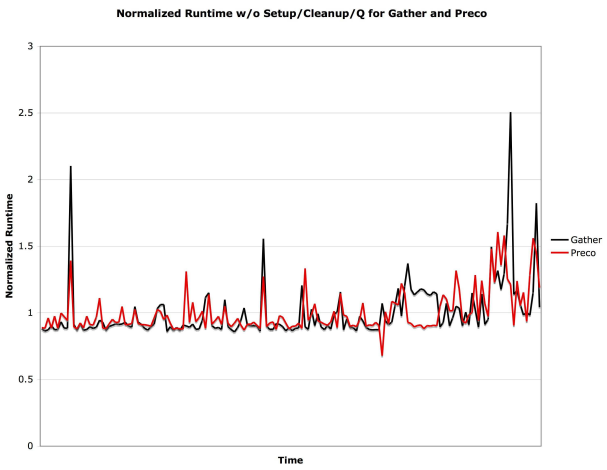

Fig. 5. Normalized Gather and PreCo measurements (excluding the setup, cleanup and queue wait times) vs. time.

Although all these correlation coefficients are fairly low, the trends can be seen more clearly in Figure 5, which shows the normalized runtimes of Gather and PreCo counting just the time to transfer the data file, run the computation and transfer the results file. This figure provides some evidence that a slow Gather runtime almost always corresponds to a slow PreCo runtime. In what follows we investigate whether we can use probe measurements to predict an application's performance.

A. Prediction Methods

To predict the performance of an application, we wish to predict the next point in a time series of application runtimes. However, generating the time series requires the application to run periodically, which is resource-consuming and impractical, especially for large-scale applications. (Not many people would want to regularly run their application just to be able to predict the next runtime.) To solve the problem, we try running the light-weight benchmark probes instead and propose two kinds of methods to predict the performance of applications from time series of the probe runtimes. Figure 6 provides a taxonomy starting at the root of the tree with 3 classes of methods: 1) time series prediction (TSP) which attempts to predict the next application runtime directly from a time series of previous application runtimes , 2) linear regression prediction (LRP) and 3) hybrid (TSP + LRP) methods. The latter two methods attempt to predict the application runtime from time series of probe runtimes and can exploit variable or fixed training set to train their prediction functions. Variable training sets can have different sizes. Here we only consider two training set sizes: the full history or the last 10 measurements. Methods with fixed training sets use the first x measurements to predict all following measurements. Details of all these methods are listed below.

- **Time Series Prediction (TSP)** In this method, the timings of the PreCo application are regarded as a time series and the NWS forecast tool is used to predict the performance of future runs. This uses no knowledge of the Gather probe measurements; predictions are based only based on the previous PreCo runtime history.

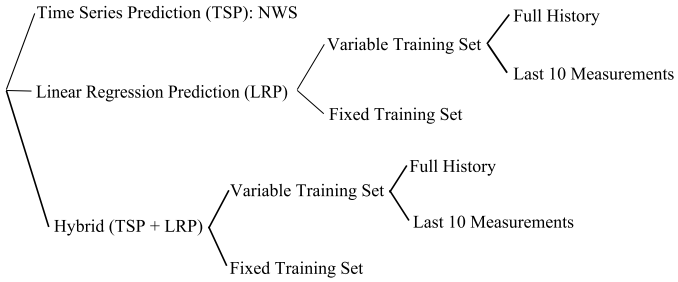

Fig. 6. Taxonomy of Prediction Methods

- **Linear Regression Prediction (LRP)** In this class of methods, we assume that the relation between the performances of Gather and PreCo is a linear function and we can manage to calculate the coefficients. Then we run Gather periodically to collect a time series and use the last measurement to first predict the next Gather runtime and then calculate the predicted runtime of PreCo. Fixed or variable length training sets (portions of the previous history time series) can be used to create the linear regression function.
  - **Variable Training Set** With this method, each prediction has its own specific training set. These sets can have different sizes. Here we only consider two situations: the full history or the last 10 measurements. When using the **full history** to make a prediction, all previous measurements of both Gather and PreCo from 1 to $n-1$ are used to predict the nth PreCo measurement. For the method using the **last 10 measurements**, we only use the last 10 measurements of Gather and PreCo to calculate the coefficients of the linear function. Either way, this method requires PreCo to run periodically with Gather; this is impractical in most real application scenarios but studied here to reveal the predictable relationship between the performances of probes and applications.
  - **Fixed Training Set** Using the fixed-training-set method, we first create the linear function using the first x measurements of both Gather and PreCo. Unlike the variable-training-set method, the linear function is not recreated for each new prediction; the linear function is used to predict all future PreCo measurements. The advantage of this method is that we do not have to regularly run the application; we only need to run the application when producing the training set, to calculate out the coefficients of the linear function; subsequently we only run the probe periodically and use that time series to make future runtime predictions for the application.

- **TSP + LRP** The above LRP methods can only be used to analyze the predictability of the application performance offline since it requires the measurement of the probe to ”predict” the performance of the application submitted at the same time. To create a practical predictor scheme, we combine the TSP and the LRP methods. First, we apply the TSP method to the time series of the probe, Gather, to predict its performance at the future time. Then the LRP method is used to predict the performance of the application, PreCo, with the previous prediction as input. The training set classification is the same as above. The hybrid method using fixed training set is a practical online prediction method without any requirement for the application to run periodically.

B. Prediction Results

For each method we computed its average relative error (i.e., absolute error divided by actual observation) when predicting all our observed application execution times. Table VII shows these relative error and coefficient of variance (CV) percentages. Some stages of the application are more predictable than others. All prediction methods predict the total runtime with a relative error of 10.55% at the worst and 8.42% at the best and a coefficient of variance of 87.17% and 90.74% respectively. The core of the application, namely, transferring the data, running the computation, and finally transferring the results, has a relative error of 11.33% at the worst and 7.92% at the best with a coefficient of variance of 98.15% and 106.65% respectively . (Note that, expectedly, all prediction methods have higher relative error for more stages with high variability.)

From the data, we can observe that the time series prediction method and the most accurate linear regression prediction method, namely, LRP with full history, have the smallest relative errors. This indicates that predicting the performance of an application by those of probes can be as good as predicting by the history of the application itself. The hybrid method with full history pays a little cost of accuracy due to its online attribute because both of its TSP and LRP parts introduce errors. But it is still almost as accurate as the previous two methods. Methods with last 10 measurements have worse accuracy, which may seem counterintuitive. It appears that ”ancient history” is of value in predicting future runs of the probe. Methods with fixed training set have the worst accuracy because the fixed linear function may not fully capture the relationship between the probe and the application. Still, with error rates of about 10%, these methods may be deemed preferable and are probably more practical given that they do not require the application to run periodically.

Overall, unless one’s grid application spends most of its time in queues (an unfortunate attribute of the system) or an inordinate amount of time staging data on shared login nodes (an unfortunate design of the application), reasonably accurate forecasts of application performance can be obtained from benchmark probe measurements.

VII. Related Work

Our work is related to many previous projects that have produced benchmarks developed for microprocessors and for High Performance Computing (HPC) systems, and more recently for to Grid computing. Benchmarks for traditional microprocessors and HPC systems fall into at least two main categories. First are low-level “probes” that measure the rates at which a machine can perform fundamental operations. Examples of this class include MAPS [10], [11], STREAM [12], the Intel MPI Benchmark [13] (formerly Pallas PMB) and SKAMPI [14] MPI benchmarks, and to some extent the LINPACK benchmark[15]. In the specific context of probes for Grid platforms, we find the GrASP project [2], which we describe and use in this work, and other projects such as RGRBench [16] that measures the performance of Grid information service and was used in [17]. The second category of benchmarks are ones designed to capture the computational needs of a class of applications. In HPC, among the best known of these are the NAS Parallel Benchmarks (NPB) [18] that are based on computational fluid-dynamics problems. Some other influential suites of this kind are SPEC [19], ParkBench [20] and SPLASH [21]. In the context of Grid computing, the NAS Grid Benchmarks (NGB) [22] have been developed as and extension to the NPB and other individual applications benchmarks have been developed such as the PreCo benchmark used in this work.

VIII. Conclusions and Future Work

A key challenge for improving the infrastructure and the operation of large-scale federated grid platforms is that of measuring and understanding resource availability and performance. In this paper we have used the Inca system in conjunction with the GrASP benchmark probes and the PreCo application to monitor two state-of-the-art grids over a 6-month time period. Our performance monitoring data has made it possible to quantify the performance variability that may be expected on such platforms and the sources of that variability. We have found that, expectedly, wait times in batch schedulers’ queues is the most relevant and prevalent source of performance variability. In terms of availability, we have experienced rather low success rates in our experiments (in the 55%-80% range), showing that these platforms are still far from having what one would consider “good” availability. We attempted to explain the sources of all failures, and found that some key failures came from the middleware infrastructure. Most of these failures should be easily correctable. Although it was not the focus of these experiments, it seems clear there is use for the probes as a diagnostic tool for platform administrators to troubleshoot the infrastructure.

Using three different prediction methods we were able to make predictions for the application’s total runtime with a relative error of 10.55% at the worst and 8.42% at the best using the different methods. Importantly, some of these methods, after some training period, rely solely on the periodic execution of light-weight benchmark probes.

TABLE VII

RELATIVE ERROR (AND COEFFICIENT OF VARIANCE) PERCENTAGES FOR DIFFERENT PREDICTION METHODS.

	Total Runtime (RT)	RT w/o Setup/ Cleanup	RT w/o Setup/ Cleanup/Q	Stage Data	Build Executable	Actual Computation	Actual Queue
Average PreCo Measurement	816.61	715.45	608.07	33.50	17.42	312.89	107.37
Standard Deviation	112.52	103.74	72.16	37.65	5.82	19.93	77.79
TSP (NWS)	8.57(90.32)	8.47(95.66)	7.92(106.65)	29.44(97.99)	3.84(165.64)	4.39(80.45)	29.37(91.05)
LRP w/ Full History	8.42(90.74)	8.73(90.71)	8.08(110.68)	27.62(84.5)	4.86(136.44)	5.83(291.95)	33.8(74.25)
LRP w/ Last 10 Measurements	9.13(78.62)	9.35(83.49)	9.44(125.48)	67.89(415.24)	6.29(293.72)	11.95(408.4)	28.31(81.85)
LRP w/ Fixed Training Set	9.96(90.61)	10.64(90.24)	10.57(92.33)	24.65(78.9)	4.55(140.17)	4.41(85.34)	37.13(76.33)
TSP+LRP w/ Full History	8.99(85.5)	9.1(87.69)	8.54(107.93)	30.92(82.14)	4.79(130.75)	4.41(75.72)	34.05(84.04)
TSP+LRP w/ Last 10 Measurements	9.14(81.84)	9.11(87.4)	8.49(97.04)	65.88(227.4)	4.27(121.68)	5.38(105.91)	29.22(83.44)
TSP+LRP w/ Fixed Training Set	10.55(87.17)	11.11(88.86)	11.33(98.15)	28.43(87.64)	4.69(133.39)	4.55(84.14)	37.84(84.64)

ACKNOWLEDGMENTS

This work was supported in part by NSF STI award number 0230925 entitled Data Intensive Grid Benchmarks and by the SciDAC award entitled High-End Computer System Performance: Science and Engineering. We would also like to thank Claudia Padula for the aid she provided with our statistical analysis.

REFERENCES

[1] "TeraGrid Inca Test Harness and Reporting Framework," http://tech.teragrid.org/inca/.

[2] G. Chun, H. Dail, H. Casanova, and A. Snavely, "Benchmark Probes for Grid Assessment," in *Proceedings of the High-Performance Grid Computing Workshop*, April 2004.

[3] "National Center for Microscopy and Imaging Research," http://ncmir.ucsd.edu/, 2005.

[4] "The TeraGrid Project," http://www.teragrid.org.

[5] A. K. Sinha, B. Ludaescher, B. Brodaric, C. Baru, D. Seber, A. Snoke, and C. Barnes, " GEON: Developing the Cyberinfrastructure for the Earth Sciences - A Workshop Report on Intrusive Igneous Rocks, Wilson Cycle and Concept Spaces," http://www.geongrid.org/workshops/conceptspace/igneous_rocks/workshop_report_intrusive_igneous_rocks.pdf, 2004.

[6] "Distributed European Infrastructure for Supercomputing Applications," http://www.deisa.org/, 2006.

[7] " NSF Middleware Initiative Release 7," http://www.nsf-middleware.org/Lists/NMIR7/AllItems.aspx.

[8] "The Inca Reporter Guide," http://tech.teragrid.org/inca/www/documentation.html.

[9] W. Smith, V. Taylor, and F. I., "Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance ," in *Source Lecture Notes In Computer Science; Proceedings of the Job Scheduling Strategies for Parallel Processing table of contents*, vol. 1659, 1999, pp. 202–219.

[10] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for application performance modeling and prediction," in *Proceedings of Supercomputing*, November 2002.

[11] "MAPS Benchmark," http://www.sdsc.edu/PMaC/MAPs, 2005.

[12] "STREAM: Measuring Sustainable Memory Bandwidth in High Performance Computers," http://www.cs.virginia.edu/stream/, 1995.

[13] "Intel MPI Benchmarks 2.3," http://www.intel.com/cd/software/products/asmo-na/eng/cluster/mpi/219848.htm, 2005.

[14] "SKaMPI-Benchmark," http://liinwww.ira.uka.de/~skampi/, 2005.

[15] "LINPACK Benchmark," http://www.top500.org/lists/linpack.php, 2005.

[16] B. Plale, C. Jacobs, Y. Liu, C. Moad, R. Parab, and P. Vaidya, "Benchmark Details of Synthetic Database Benchmark/Workload for Grid Resource Information," Dept. of Computer Science, University of Indiana, Tech. Rep. TR-583, August 2003.

[17] D. Kodeboyina and B. Plale, "Experiences with OGSA-DAI: Portlet Access and Benchmark," in *Global Grid Forum Workshop on Designing and Building Grid Services*, September 2003.

[18] D. H. Bailey and E. Barszcz and J. T. Barton and D. S. Browning and R. L. Carter and D. Dagum and R. A. Fatoohi and P. O. Frederickson and T. A. Lasinski and R. S. Schreiber and H. D. Simon and V. Venkatakrishnan and S. K. Weeratunga, "The NAS Parallel Benchmarks," *The International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[19] "The SPEC Benchmarks," http://www.specbench.org.

[20] T. Hey and D. Lancaster, "The development of Parkbench and performance prediction," *The International Journal of High Performance Computing Applications*, vol. 14, no. 3, pp. 205–215, 2000.

[21] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and methodological considerations," in *Proceedings of the 22nd International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June 1995.

[22] M. Frumkin and R. Van der Wijngaart, "NAS Grid Benchmarks: A Tool for Grid Space Exploration." *Cluster Computing*, vol. 5, no. 3, 2002.