GIPSE: A Toolset for Streamlining the Management Aspects of the Grid for Simulation-based Research

A. Striegel, M. Shorts, E. Stuntebeck, D. Salyers, J. A. Izaguirre
Dept. of Computer Science & Engineering
University of Notre Dame
Notre Dame, IN 46556 USA
E-mail: *striegel,mshorts,estunteb,dsalyers,izaguirr@nd.edu*

***Abstract*— Although the grid allows the researcher to tap a vast amount of resources, the complexity involved in utilizing this power can make it unwieldy and time-consuming. The GIPSE (Grid Interface for Parameter-driven Simulation Environments) toolset aims to solve this issue by freeing users from scripting, debugging, and other minutia involved in managing simulations on the grid. GIPSE, which interacts seamlessly with existing grid software (Globus, etc.), abstracts interactions with the grid to present a research-centric view of the process rather than the typical task-centric view. GIPSE offers an alternative interface to the grid that removes the need for application specific wrappers around parameter-driven simulations.**

In this paper, we discuss how GIPSE bridges a critical gap between existing tools and management of the overarching data result. Furthermore, we outline the unique aspects of GIPSE, including an overlying XML data flow for result management, intuitive user interactions for controlling simulations, increased accuracy for task estimation, deadline scheduling, and the ability to provide data-driven dynamic task submissions. We conclude with discussions of future extensions to GIPSE.

Keywords: *Grid computing, task management, middleware*

I. Introduction

Over the last few years, the notion of grid computing has emerged as a promising approach for harnessing computing power on a previously unheralded scale. The foundational work of the Globus project [1] and others [2]–[6] offer a gateway for users to both contribute to and access the ever growing collection of grid resources. With such great resources comes, however, also a torrent of data that must be managed with regards to both creation (task construction) and output (analysis, storage, etc.). While the grid provides the necessary tools for locating and utilizing resources, the grid is targeted towards managing individual tasks rather than the overarching result. This is only natural, as the purpose of the grid is completing tasks, not management of data. Hence, the tasks related to management, such as splitting up tasks, submitting tasks to the grid, parsing the results, and analyzing the results are left to the user (see Figure 1).

While various toolsets such as the Java CoG kits [7] and others [8] have reduced the complexity associated with interfacing with the grid, supporting interfaces must still be developed to bridge the final interface to the application. Although the need for a vertical stove-pipe solution has been removed, the interface presented to the user is still task-oriented. For parameter-driven simulation environments with large parameter spaces, the development gap between the existing tools and the actual results can be quite significant (see Figure 1).

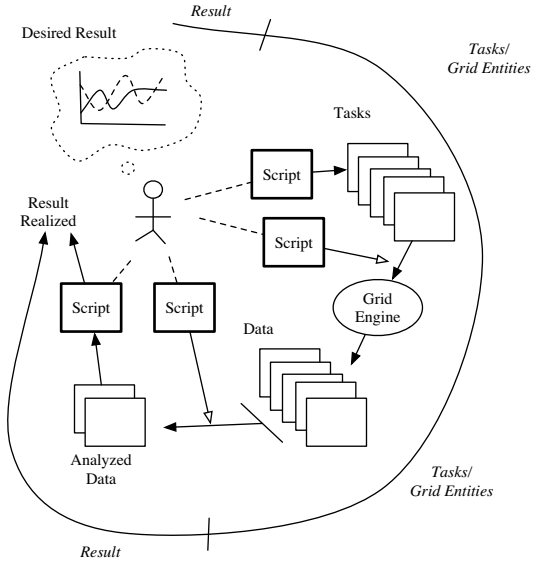

Fig. 1. Current process with existing toolset

To that end, we propose the concept of a generic framework for parameter-driven simulations that links the entire data production process (task creation, task submission, data parsing, and data storage) in a result-centric fashion. The Grid Interface for Parameter-driven Simulation Environments (GIPSE) toolset offers an alternative interface to the grid that is specifically tailored to simulation-based research. In short, GIPSE sits on top of existing grid tools (Globus, Java CoG) and employs XML metadata to link the serivce-centric nature of the grid with the data-oriented nature of the researcher. GIPSE does not change the execution of the grid but rather transforms the view of the grid through the use of the XML metadata (see Figure 2). Thus, the researcher can still interact with the grid using traditional tools in a task-centric view with GIPSE-linked tasks simply appearing as normal tasks.

Beyond the immediate time savings due to the streamlined interface, GIPSE impacts the entire simulation process.

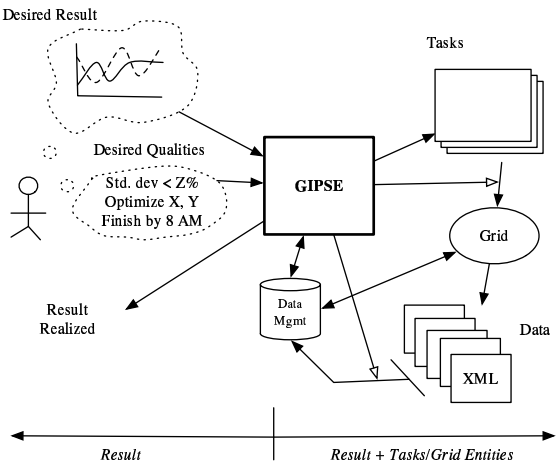

Fig. 2. Improving the process with GIPSE

Whether it is abstracting storage to allow for SQL-like queries on the basis of individual data fields or reliability enhancements due to built-in sanity checking and simplified regression testing, GIPSE provides immediate benefits as well as advanced functionality. Above all, GIPSE frees the researcher to work from a higher level while still leveraging the vast array of grid technologies currently available.

The remainder of our paper is organized as follows. In Section II, we motivate GIPSE through examples from the domains of network simulation and bio-complexity. Next, in Section III, we describe the basic components and benefits of GIPSE. Section IV continues by analyzing the components of GIPSE in more detail. Finally, Section V offers several concluding remarks.

II. Case Studies: Network Simulation and Bio-Complexity

A. Network Simulation

To better motivate the need for GIPSE, we first describe an application scenario in the realm of network simulation. Given the fact that the monitored behavior must be verified over a wide variety of environments through different network topologies, background traffic, and client heterogeneity, the process of network simulation lends itself quite well to grid computing. However, as will be shown shortly, the gap between where the existing set of tools for grid computing end and the actual required interfaces for utilizing the grid can be quite significant and time consuming to address.

Table I summarizes the characteristics associated with a typical data run (final results) associated with a typical data run. For each step along the process, scripts and user interactions were employed to create the simulation results that are summarized below:

1. A Perl script generated all of the tasks for the grid by appropriately varying the arguments to the simulation executable. The data output file argument was mapped to denote the key parameters (Foo-X-Y-Z.csv).

Characteristic	Network Simulation	Bio-complexity
Application	ns-2 [9]	PROTOMOL [10]
Research area	Content distribution	Bio-molecular modeling
Avg. Task Length (min) (Range)	30 5-240	7200 (4k-20k)
Arguments	35+	15+
Output Fields	100+	10+
Parameter Sweeps	7-12	10
Tasks / Data Run	1-1.5k	1-2k
Generated Data per Run	200-500MB	10-100GB

TABLE I
Case Study Characteristics

2. The tasks were submitted to the grid using the same Perl script and grid input files generated in the previous step.
3. The grid was monitored by using command-line interactions and the completion time was roughly estimated by extrapolating the previous throughput.
4. A final collection of scripts took the output data files, consolidated them into related groupings (all Foo-* files together), analyzed the file for data output, and created the output graphs.
5. The final results were stored through shell commands and in an ad hoc manner through multiple sub-directories.
6. If necessary, multiple data runs were executed to correctly position the selection of parameter sweep and to repair any coding issues (scripting, simulation, etc.).

While the above scripts were sufficient to allow for the use of grid functionality, the scripts suffered from several key weaknesses:

- *Task estimation:* Although the use of task run-time extrapolation (total time vs. tasks completed) did allow for a rough estimated completion time, the results were often skewed (to be lower than actually required) due to the heterogeneity of task execution length.
- *Redundant executions:* In order to keep the scripts simple, *for* loops were used to create the necessary tasks for the various data points. As a result, redundant tasks were allowed to creep into the simulation.
- *User steerability:* Unless the tasks were appropriately organized in advance for individual tests, it was difficult to remove complete dataset tasks to move up the completion time (i.e. reduce the number of environments from 4 to 3). Since the grid engine knows only tasks (IDs), one must rely on the current queue of the grid engine or *a priori* organization to then determine which tasks to remove. This problem is only further compounded in the event of a busy grid (i.e. simulation tasks interspersed with other users' tasks).
- *Dataset management:* The use of scripts and wildcards (for simplicity) does not lend itself to easily mitigating the effect of incomplete datasets. In the event that the user does steer the results, partial datasets must be purged manually or through the development of additional scripts.

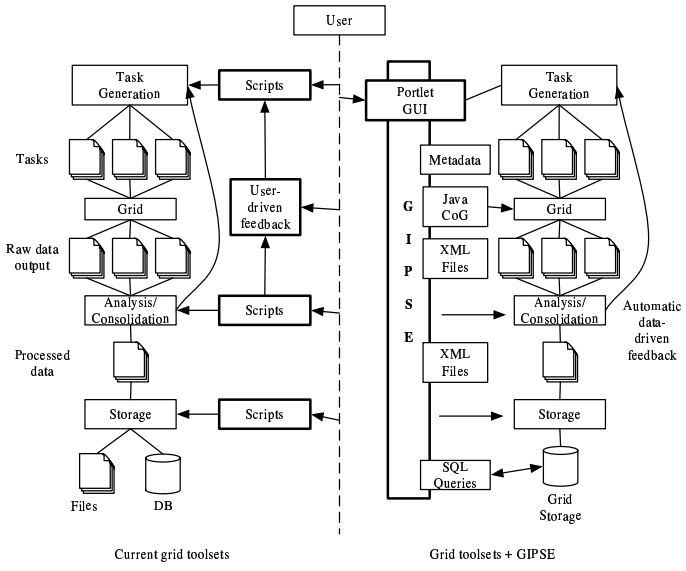

Fig. 3. GIPSE vs. existing tools

While this list is not necessarily complete, it illustrates the point that management of the simulation process can be a significant burden to the researcher for even the most basic of functionality.

B. The GIPSE Difference

In contrast, GIPSE offers a distinctly different perspective for the simulation. To start, all of the typically necessary Perl/shell scripts for submitting tasks to the grid or parsing the data are removed. The user can choose between an XML-driven interface or a fully featured GUI (available as a portlet) for defining the simulation and the desired outputs.

The interface to the simulation is governed by a list of arguments that are based on a list of user-defined variables. In fact, GIPSE shares many of the properties of scripts (variables, etc.). However, GIPSE removes the minutia of script programming (debugging, error checking) in a simple, intuitive interface. Furthermore, unlike the previous case with multiple *disjoint* scripts, the definitions of variables and arguments are used throughout the entire simulation process.

The list of data fields is gleaned from a single benchmark execution of the simulator from the converted data file via XML. The remainder of the simulation setup is completed by defining configurations (groups of fixed settings for variables) that can be used to compare the different approaches.

Once the user has defined the basic interface points for the simulation (what can be varied), the user works from the perspective of the final graph, not tasks or scripts. The graph selection process is further streamlined as the names of what is available to vary and plot are already available from the earlier setup. Thus, the user selects what to vary for the x-axis (a variable that changes and impacts the arguments to the simulation), the configurations needed (approaches to compare), and what values to plot (the actual data). For the researcher, the setup process now deals with the desired end results (see Figure 3).

Upon submission of a simulation suite, GIPSE automatically creates the underlying scripts and interacts with the chosen grid interface (Globus via JavaCoG [7], Condor-G [11], etc.). In contrast to the earlier case where the user could only indirectly watch the progress of the simulation (based on the queue and what has been completed), GIPSE presents a unified point for watching the progress from a simulation perspective (bar graphs, breakdowns for each test, completion times, etc.). In addition, the user can change the computational requirements of the overall simulation with a single click (i.e. reduce the averaged environments from 4 to 3), leaving GIPSE to handle the complexity of using only complete datasets when the process is complete.

Furthermore, GIPSE can assess a much more accurate estimated completion time. Beyond using historical or benchmarked runs, GIPSE can infer changes in computational length due to changes in inputs (arguments) since GIPSE knows under what parameters a task was generated. Thus, GIPSE can offer an effective deadline driven service[1] driven by actual results, not by user guesswork. Finally, GIPSE assists with the simulation process by automatically parsing and analyzing the results and producing the desired graphs. The data storage is handled by GIPSE with the final data (plus any desired statistical analysis) presented to the user with no additional intervention or scripting required.

C. Advanced Case Study: Bio-Complexity

For a second example, we consider the field of bio-complexity. The advent of massive amounts of biological information brought by the human-genome and associated projects requires ever increasing amounts of computational power and storage to try to make sense out of this data. One of the key problems is understanding the preferred geometric conformations of proteins and other molecules, since these dictate the biological function they are able to perform.

Hence, different sampling methods are being investigated to discover conformations of biomolecules. Due to the fact that molecules have hundreds of important degrees of freedom and multiple states in which each can reside, one needs smarter ways of generating samples beyond naive, combinatorial approaches. For instance, various works have been pursued employing the simulation of molecular dynamics with stochastic Monte Carlo steps (Hybrid Monte Carlo or HMC), cf. [12]–[16]. With its inherent complexity, many optimizations to the method are possible, and such optimizations typically involve the tuning of several parameters. The process of automatic optimization of algorithms and run-time software for the above cases requires extensive testing of the algorithm space and encapsulation of rules for guiding the selection of algorithms or parameters.

In general, the method developer needs to characterize the parameter space (e.g., this is a continuous parameter in the

[1]For instance, maximize the number of complete datasets by 8 AM tomorrow for a conference paper due at 5 PM.

range 0 to 1), and the performance metrics that will be monitored. For example, in the sampling problem described above, the cost per new conformation discovered is the primary performance metric, and secondarily, the variance in averages that are computed at the proposed conformations. The aim of the method developer is to lower both metrics with respect to previous methods.

D. GIPSE Performance Contracts

Certain choices of method parameters quickly lead to unacceptable accuracy (e.g., variance is too high) or excessive time complexity. In GIPSE, the algorithm developer specifies performance targets for her algorithm through *performance contracts*. The algorithm design can be made by iteratively refining its contracts. Design by contract is a technique that originates in the area of software engineering, cf. [17].

A performance contract specifies preconditions that have to be met by the user of the algorithm, and gives guarantees on what the algorithm delivers. For example, given that a user passes non-negative arguments to a real-valued square root function, the algorithm guarantees a correct result. In case that either the precondition or postcondition is violated, there is a violation of the contract and an exception or signal is raised. The algorithm developer can specify how to handle that exception, by either terminating the execution and indicating what the violation was, or by changing the parameters and retrying.

The primary benefit of GIPSE in this application is to aid in the evaluation of algorithms and in the fine tuning of its parameters. The performance contract serves a dual purpose in GIPSE:

- Discover a range of parameters for which the algorithm performs well. For example, by recording those parameter combinations for which the algorithm performs badly, as indicated by the exceptions.
- Guide and optimize the use of the grid computational resources, by terminating or eliminating simulations with ranges of parameters that are known to yield bad results. This may be seen as an adaptive pruning procedure for searching in the parameter space of an algorithm.

E. Previous Work

For many solutions employing grid technologies, the solution has been to develop a vertical 'stove-pipe' solution [18]–[20]. With the introduction of tools such as Java CoG [7] at the lower level and grid-oriented portals at the higher level [8], the need for the complete vertical solution has been removed. Other tools such as LSF and PBS assist with batch functionality (helping only marginally with task generation) but still deliver a similar process such as the in the pre-GIPSE case studies. Hence, a critical gap still exists between the service-oriented nature of the grid and the result-oriented focus of the end researcher.

Specifically, GIPSE operates as a middleware between the grid service interface to provide an alternative interface to the user tailored to the parameter-driven simulation environment.

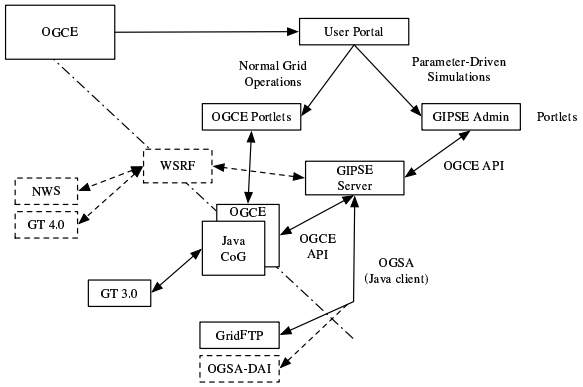

Fig. 4. GIPSE Toolset

GIPSE does not replace existing grid tools but rather focuses on the issues associated with parameter-driven simulations to reduce setup and operation to only minor configuration overhead. To the best of our knowledge, GIPSE is the first tool to provide reactive data-aware functionality in an application-agnostic package for simulation-oriented research.

III. GIPSE Toolset Overview

We begin the overview of GIPSE by showing the basic interactions of the GIPSE toolset in Figure 4 with current grid tools. Due to its structure, GIPSE offers several benefits that should be noted:

- *Common data format:* With its XML-based definitions for basic data inter-operability, GIPSE offers a consistent and open interface for storing, analyzing, and gathering data. Furthermore, the XML format allows for the simple inclusion of extensions. The format allows GIPSE to provide a placeholder in the data process while providing a basic level of inter-operability.
- *Simulation-wide namespace:* Rather than dealing with proprietary or position-dependent data output (argument 5, column 6), GIPSE allows the user to employ intuitive strings for identifying the significance of individual fields/values. The inclusion of a simulation-wide namespace has implications for not only ease of use and correctness but also for advanced features such as regression testing.
- *Intelligent task submission:* Due to the fact that GIPSE generates the tasks that are submitted to the grid, GIPSE can intelligently eliminate the submission of redundant executions to the grid. In the event that multiple graphs share data points, GIPSE will condense the task set to the minimum required and correctly fill the datasets for the final results.
- *Data-aware management:* Since GIPSE can understand and parse the underlying data (due to the base XML data format), GIPSE can offer a host of features not previously available on a generic basis. The ability to be aware of the data has implications for not only data mining but

also for dynamically driven simulations (e.g. simulation until condition X is satisfied across the data).

- *Dynamically-driven simulations:* GIPSE can dynamically alter the requested workflow to meet conditions specified by the user. For instance, GIPSE can provide deadline-based optimizations while still maintaining completeness of the datasets. Furthermore, GIPSE can provide statistically-driven simulations (run until the standard deviation is X) or intelligently steered simulations (manipulate X, Y until a result of Z is achieved) , a significant improvement over the cycle of user guesswork from before.

IV. GIPSE Components

A. XML Data Format

For most grid-friendly applications, solutions that are aware of the underlying data are application-specific. In the absence of a standardized base format, the tools can become time consuming to develop and maintain. In contrast, the foundation for the basic and advanced features of GIPSE begins with the use of a standardized XML format that is not application-specific (see Figure 5).

When data is produced by an individual simulation run, the data is directly output to or converted to the base GIPSE XML specification[2]. In addition to the data itself, the file is tagged with identifiers from the GIPSE server (Task ID, Simulation Name, etc.) and with information regarding the execution of the individual run (performance, system specs, etc.). Furthermore, extensions can be included in the XML output that are not processed by GIPSE directly but are still stored by GIPSE for third party extensions.

While the use of XML by GIPSE does introduce inefficiency with regards to intermediate storage and parsing, we believe the inter-operability features of a base XML specification outweigh the negative aspects. Furthermore, GIPSE does not preclude the use of a proprietary format for efficiency as such modules can be incorporated as custom extensions for GIPSE. Rather, the key point of the standardized XML file (via well-known tags) is to provide a basic level of inter-operability.

The actual conversion process to the GIPSE XML format can be accomplished in one of two fashions. First, a GIPSE library (available in both Java and C++) can be directly included in the simulation and used for statistical data gathering (or summary). In this case, the inter-operability with GIPSE is seamless as the library contains appropriate output and submission functionality. In the second option, the user writes a client that incorporates the GIPSE library (for output and submission) to convert the customized format to GIPSE. The client case is ideal when efficient methods already exist in the simulation or the GIPSE library would be difficult to include in the simulation directly.

[2]Due to space requirements, the GIPSE XML specification is not included in the paper.

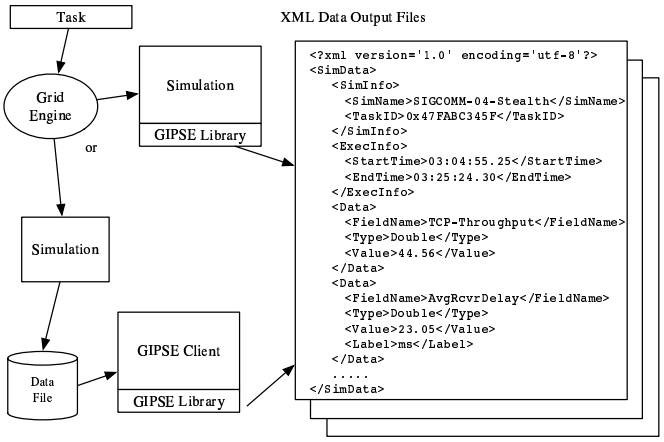

Fig. 5. Example GIPSE XML Output

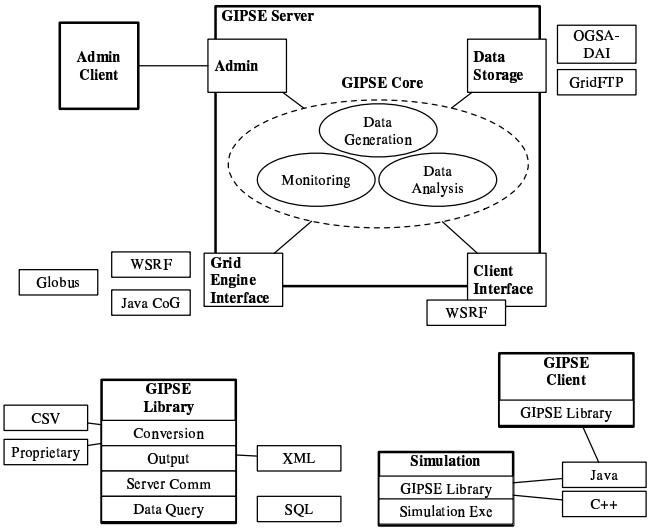

Fig. 6. GIPSE Toolset Architecture

B. GIPSE Server via Web Services

The GIPSE server acts as the central clearinghouse for all GIPSE-related operations. The GIPSE server is responsible for providing the control and interactions of simulation-related tasks to the underlying grid computing engine. Beyond the interface points of the grid (administrative client, data storage, grid engine, data client), the core of the server can be divided into three main components (see Figure 6):

- *Data Generation:* The data generation component controls how tasks are generated for submission to the grid. In addition to acting on dynamic conditions (deadlines, statistics, performance contracts, etc.), the task generation component is responsible for constructing the list of tasks for a given simulation.
- *Monitoring:* The monitoring component interfaces with the grid engine interface to derive the current status of simulations being executed on the grid. The monitoring routine is responsible for distilling the task-level perfor-

mance into simulation-level performance readings.

- *Data Analysis:* As data is returned from the various tasks to the GIPSE server, the data analysis module is responsible for parsing the data (in the GIPSE XML format) and updating the appropriate status for the simulation data runs. In addition, the data analysis module runs any requested statistical analyses on the returned data and feeds the status into the task generation component (if necessary).

Data Generation: Whereas the workload for task generation with current grid toolsets is left to the user and typically done in an application-manner, GIPSE handles all aspects of task generation in an application-agnostic manner. Due to the fact that GIPSE is targeted towards parameter-driven simulations, an interface is presented that lets the user select which variables to alter for the simulation. Based on the selected sweeps, the GIPSE server will generate the appropriate tasks and submit those tasks to the grid. GIPSE then adds the associated metadata (specific file naming conventions, etc.) to link the progress of individual tasks to the overall progress of the data run for the simulation. If necessary, a single task may be made up of a complete workflow specified by tools such as GridAnt [21].

In addition, GIPSE further refines the process associated with task submission. Rather than simply trusting the correct operation of the user operation, GIPSE can enforce sanity checking across both arguments and variable settings for the entire process. The corrections to the process can be applied with GIPSE automatically applying sanity checking rather than the user having to build such functionality into a script. Furthermore, the sanity checking is not limited to task generation as further verification can be applied the data which is discussed in more detail below.

Task Monitoring: One of the simplest ways to estimate task completion is to simply extrapolate the average of past results to determine an estimated completion time. For tasks whose computation time is independent of the input parameters or whose previous performance history has been quite steady, the estimation can be fairly exact. Assuming a homogeneous computational grid (for simplicity), an approximate formula for determining a data run's completion time via extrapolation can be computed as follows:

$$\frac{\sum_{i=0}^{C} RT(t_i)}{C} \times Q - \sum_{i=0}^{Q} RT(t_i)$$

where C is the number of completed tasks, $RT(t_i)$ is the run-time of task t_i, and Q is the currently number of queued tasks. However, as shown in Figure 7, the estimation tends can be have a significant amount of error. Figure 7 plots the task estimation for a single parameter sweep from historical data from the first case study. Most notably, the fact that GIPSE is aware of the inputs that contributed to a specific task and its respective performance allows GIPSE to more quickly arrive at the correct estimation for data run completion.

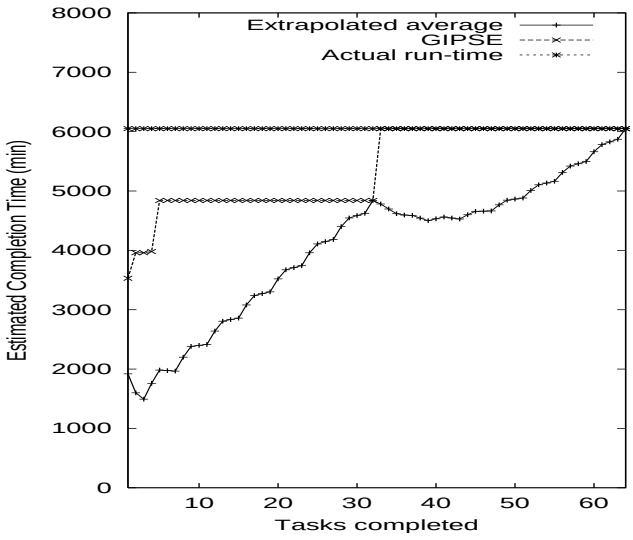

Fig. 7. Comparison of task estimation approaches

GIPSE calculates the estimated performance of future tasks by including a weighting of the relative impact of parameters rather than a blind average as listed below:

$$\sum_{i=0}^{Q} RW(t_i, P_i) \times B - \sum_{i=0}^{Q} RT(t_i)$$

where $RW(t_i)$ is the relative weight of task t_i using parameters P_i versus the base computation time B. While this estimation does not account for other tasks on the grid, it offers a more precise estimate of the amount of computational time required for a specific set of parameter-driven simulations. By coupling this improved estimation with historical results (for the individual tasks) and appropriate weightings from the heterogeneous grid, GIPSE can allow the researcher to react quicker to the underlying impact of the computation. In the cases of parameter exploration or parameter optimization, the improved reaction time can be a significant benefit to the researcher.

A secondary application of the increased accuracy of task estimation would be better scheduling and run-time bounding with heterogeneous resource requirements. Using existing toolsets, an absolute bound can only be used if it satisfies all tasks in the data run. The same type of problem applies to historic data or deviations from the average if the run-time is sufficiently heterogeneous with regards to parameter input. Without sufficient granularity with regards to individual task resource consumption, the larger tasks will always dominate leaving the inaccuracy with regards to smaller tasks simply ignored. Since GIPSE understands the relative impact of parameters on resource consumption, it can better monitor tasks for run-time deviations and schedule tasks with more precise resource requirements. Thus, GIPSE is able to capitalize on its result-awareness to deliver finer grained resource accuracy with regards to resource requirements for utilizing the grid.

Data Analysis: Out of the three components to the GIPSE server, the data analysis component offers the most dramatic shift in terms of current grid toolsets. Whereas data-awareness would typically require an application-specific solution, the XML basis for data transmission allows the GIPSE server to be data-aware without application-awareness. Furthermore, by virtue of GIPSE understanding the context in which the data was produced, considerable functionality can be built on top of the underlying GIPSE foundation for dynamically data-driven simulation environments.

At the simplest level, the data analysis component allows for sanity checking and verification of the resulting data output. The data itself can be further processed by GIPSE to produce relational data (without requiring a complete workflow diagram) or can be used to conduct regression testing and trending versus archived data output.

At a more advanced level, GIPSE can be used to dynamically direct the simulation for data-driven operations. For instance, GIPSE could continue to submit data runs until a specific statistical property is met (run for confidence interval of X between plots A and B). The data-driven operations can be extended to perform optimizations of output using machine learning, neural networks, or other search techniques. Furthermore, the fact that GIPSE can parse the data to a generic format allows for new modules to be easily incorporating using web services interactions with GIPSE.

C. GIPSE Data Repository

The final portion of GIPSE lies with the GIPSE data repository. The GIPSE data repository completes the entire abstraction of management for grid computing by virtualizing how the resulting data is stored on the system. In the absence of a virtualized storage system (for instance an individual user submitting tasks and storing results on their account), the storage and future retrieval of the data can be almost as problematic as creating the data in the first place. In contrast, GIPSE offers two important qualities, namely organization (how to find the correct data out of the numerous datasets) and traceability (the impetus for creating the data).

GIPSE simplifies the entire process by taking care of all aspects of storage. To the user, it simply appears that GIPSE is a giant database holding all of the various data iterations that have been produced by the simulations. In practice, the data itself may be stored in a variety of fashions (centralized DB, grid storage via OGSA-DAI [22], etc.). For cases such as bio-complexity where the large amounts of data are distributed amongst thousands if not millions of files, this abstraction is extremely powerful.

V. Summary

In summary, GIPSE offers a new approach for the management of simulation-oriented grid computations. Rather than fundamentally changing the underlying grid engines, GIPSE abstracts the interactions with the grid engine to present a research-centric view of computation rather than exposing the service-centric view. Table II presents a summary of the numerous improvements offered by GIPSE. Through its flexible XML-based interactions (configuration and data organization), the GIPSE package removes the complexity of managing grid software and offers the potential to dramatically lower the barriers to simulation-oriented research taking greater advantage of grid computing. Thus, we feel that GIPSE offers significant benefit and is a compelling platform for future development.

Feature	Existing Tools	GIPSE
Emphasis	Tasks	Overarching result
Scripting	Yes, Many	None
Data Knowledge	App-specific	General
Efficiency	With user optimization	Automatic
Deadline-Driven	User guestimation	Built-in
Data-Driven (Statistics)	No, per-app support	Built-in
User Steerable	Difficult, must organize tasks correctly	Automatic
Database queries	App-specific	Data client, XML
Inter-operability	None	XML specification
Performance Contract	None, script	Built-in, modular
Data Management	Directory or app-specific	Built-in, modular
Regression Testing	Difficult, scripting	Built-in

TABLE II
SUMMARY - GIPSE TOOLSET IMPROVEMENTS

Our future work includes further refining the GIPSE framework to operate under the upcoming WSRF framework, the development of various components for parameter exploration, and the incorporation of support for existing workflow managers.

Acknowledgements

The authors would like to thank Gautam Shewakramani, Brian Bien, Kevin McCusker, and Christopher Picardo for assisting with the initial work on the GIPSE prototype toolset.

References

[1] "Globus toolkit," Available at www.globus.org.
[2] "Condor software package," Available at www.cs.wisc.edu/condor.
[3] "Sun ONE grid engine," Available at wwws.sun.com/software/gridware.
[4] "Nimrod software package," Available at www.csse.monash.edu/ davida/nimrod.
[5] "LEGION grid engine," Available at legion.virginia.edu.
[6] "Open grid services architecture," Available at www.globus.org/ogsa.
[7] Gregor von Laszewski, Jarek Gawor, Peter Lane, Nell Rehn, Mike Russell, and Keith Jackson, "Features of the Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, accepted.
[8] D. Gannon, G. Fox, M. Pierce, B. Plale, G. von Laszewski, C. Severance, J. Hardin, J. Alameda and M. Thomas, and J. Boisseau, "Grid portals: A scientist's access point for grid services (draft 1)," *Global Grid Forum*, Sept. 2003, Work in Progress.
[9] "UCB/LBNL/VINT Network Simulator - ns (version 2)," Available at www.mash.cs.berkeley.edu/ns/.
[10] T. Matthey, A. N. Ko, and J. A. Izaguirre, "ProtoMol: a molecular dynamics research framework for algorithmic development," in *SpringerVerlag LNCS 2659, Computational Science-ICCS 2003*, Melbourne, Australia and St. Petersburg, Russia, June 2003, pp. 50–59.
[11] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: a computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, pp. 237–246, 2002.
[12] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, "Hybrid Monte Carlo," *Phys. Lett. B*, vol. 195, pp. 216–222, 1987.

[13] Jesús A. Izaguirre, Scott Hampton, and Thierry Matthey, "Shadow Hybrid Monte Carlo: An improved method for conformational sampling of biomolecules," Submitted to *Journal of Computational Physics*, 2003.

[14] S. Hampton and J. A. Izaguirre, "Linearly scalable hybrid monte carlo method for conformational sampling of large biomolecules," in *SIAM Conference on Computational Science and Engineering*, San Diego, CA, Feb. 2003, SIAM, Preprint available at `http://www.nd.edu/~izaguirr/papers/SAC03_HaIz0x.pdf`.

[15] Q. Ma and J. A. Izaguirre, "Long time step molecular dynamics using targeted langevin stabilization," in *Proceedings of the ACM Symposium on Applied Computing, Melbourne, FL*, New York, 2003, pp. 178–182, ACM.

[16] George S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer Series in Operations Research. Springer-Verlag, New York, 2000.

[17] B. Meyer, *Object-Oriented Software Construction*, Prentice-Hall, 2nd edition, 1997.

[18] M. Bhandarkar, G. Budescu, W. F. Humphrey, J. A. Izaguirre, S. Izrailev, L. V. Kal, D. Kosztin, F. Molnar, J. C. Phillips, and Klaus Schulten, "BioCoRE: a collaboratory for structural biology," in *Proc. of the SCS International Conference on Web-Based Modeling and Simulation*, San Francisco, California, 1999, pp. 242–251.

[19] K. Keahey et. al, "Computational grids in action: The national fusion collaboratory," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1005–1015, Oct. 2002.

[20] M. Russel et. al, "The astrophysics simulation collaboratory: A science portal enabling community software development," *Cluster Computing*, vol. 5, pp. 297–304, 2002.

[21] "Gridant-client-side workflow management with ant," July 2002, Whitepaper.

[22] "Ogsa-dai: Open grid services architecture - data access and integration," Available at www.ogsadai.org.uk.