The Virtual Instrument: Support for Grid-enabled MCell Simulations

Henri Casanova [1,2] Thomas Bartol [3] Francine Berman [1,2]
Adam Birnbaum [1] Jack Dongarra [4] Mark Ellisman [5]
Marcio Faerman [2] Erhan Gokcay 3 Michelle Miller [4]
Graziano Obertelli [6] Stuart Pomerantz [7] Terry Sejnowski [3]
Joel Stiles [7] Rich Wolski [6]

[1] San Diego Supercomputer Center
[2] Dept. of Computer Science and Engineering, University of California, San Diego
[3] Computational Neurobiology Laboratory, Salk Institute
[4] Dept. of Computer Science, University of Tennessee, Knoxville
[5] National Center for Microscopy and Imaging Research,
University of California, San Diego
[6] Dept. of Computer Science, University of California, Santa Barbara
[7] Pittsburgh Supercomputer Center

Ensembles of widely distributed, heterogeneous resources, or *Grids*, have emerged as popular platforms for large-scale scientific applications. This paper presents the *Virtual Instrument* project, which provides an integrated application execution environment that enables end-users to run and interact with running scientific simulations on Grids. This work is performed in the specific context of MCell, a computational biology application. While MCell provides the basis for running simulations, its capabilities are currently limited in terms of scale, ease-of-use, and interactivity. These limitations preclude usage scenarios that are critical for scientific advances. Our goal is to create a scientific "Virtual Instrument" from MCell by allowing its users to transparently access Grid resources while being able to steer running simulations. In this paper, we motivate the Virtual Instrument project and discuss a number of relevant issues and accomplishments in the area of Grid software development and application scheduling. We then describe our software design and report on the current implementation. We verify and evaluate our design via experiments with MCell on a real-world Grid testbed.

1. INTRODUCTION

Grids [24, 27] are large collections of resources (computational devices, networks, on-line instruments, storage archives, etc.) distributed over the wide-area and have enormous aggregate potential and have become popular platforms for running large-scale, resource-intensive applications. Many challenges are to be addressed in order to provide the necessary mechanisms for discovering, accessing, monitoring, and aggregating Grid resources. Consequently, a large effort has been made and is still underway to provide middleware technology as a base Grid software infrastructure [28, 27, 25]. However, although middleware provides fundamental building blocks it is not designed to be used directly by Grid users. Instead, Grid application-level tools must be provided with the goal of both building new and higher-level functionality on top of base Grid services, as well as hiding the complexity of the Grid from the end-user. One approach is to provide programming models that implement high-level abstractions for building Grid applications [41, 37, 59, 10], or even general purpose Grid application development environments [8]. Another approach is to implement execution environments in which a user can "drop" his/her application for Grid execution while maintaining the convenience and the illusion of a desktop execution. Such environments include Grid portals, which have been implemented successfully for many scientific applications and provide users with a familiar Web browser interface to launch and monitors application runs on Grid resources [29, 56, 3]. As a result, several efforts have provided toolkits to help in the development of Grid portals [43, 57, 55, 32].

This work was supported by the National Science Foundation under Award ACI-0086092.

Another solution is to build integrated software environments targeted to specific applications or classes of applications. The work described in this paper belongs in this last category; our ultimate goal is to build software that makes an application behave as a scientific *Virtual Instrument* (VI) that the user can easily configure, observe, and dynamically control. We note that our work could be integrated as part of a Grid portal with minimal effort.

This work is performed in the specific context of the MCell application [40, 39, 54, 51]. MCell is a computational biology simulation framework that is used by neuroscientists to study diffusion and chemical reactions of molecules in living organisms. A single execution instance of MCell consists of multiple simulations, each producing output that is then analyzed by neuroscientists *en masse*. Although MCell provides the core functionality for running simulations, its capabilities are currently limited in terms of scale, ease-of-use, and interactivity. For example, scientists often observe interesting phenomena that emerge in the middle of an MCell run. If MCell could be re-directed to concentrate on these phenomena while executing, a great deal of time could be saved. This and other limitations preclude usage scenarios that are critical for scientific advances. The goal of the VI project is to alleviate most of the limitations of MCell usage and to provide an integrated Grid application execution environment for MCell users. This environment should provide transparency for access to the Grid as well as computational steering capabilities.

In this paper we describe our accomplishments in terms of software design and development for the Grid. This development primarily entails the realization of the VI software, a complete runtime system that supports steerable MCell executions. We present experimental results obtained when running the application on a Grid testbed and draw conclusions about the effectiveness of the VI implementation and design.

This paper is organized as follows. In Section 2 we introduce MCell and highlight specific limitations of its current usage scenarios. In Section 3 we motivate the VI project, highlight issues of application scheduling and Grid software development, and describe the VI software design and implementation, which we verify experimentally and discuss in Section 4. Section 5 discusses related work and Section 6 concludes the paper with future directions.

2. MOLECULAR BIOLOGY SIMULATIONS WITH MCELL

2.1. MCell Overview

MCell [40, 39, 54, 51] uses Monte Carlo algorithms to simulate simultaneous diffusion and chemical reactions of molecules in complex 3-D spaces. Highly realistic reconstructions of cellular or subcellular boundaries can be used to define 3D diffusion spaces, which can then be populated with different molecules [52]. Such molecules might react with others that are released periodically from different locations within the structure, to simulate the production of biological signals. The diffusing molecules move according to a 3-D random walk based on a Brownian motion model. Possible reaction events, such as binding and unbinding, are tested on a molecule-by-molecule basis using random numbers and Monte Carlo probability values. The advantages and significance of this approach are detailed in [50].

In essence, computational modeling with MCell encompasses 4 steps, each of which can require considerable computing resources:

1. **Surface design or reconstruction** – In simple cases, a set of planes might be used to define diffusion boundaries. In complex cases, cell membranes can be reconstructed as tessellated meshes from electron microscope data, and may contain on the order of 10^6 triangles.

2. **Model visualization and design** – Different types of molecules must be added to the surfaces and spaces according to realistic biological distributions and densities. The total number of molecules is highly variable but can easily reach or exceed 10^6 even for a surface area or reaction volume much smaller than a single cell.

3. **Simulation** – This step involves repeatedly running MCell with varied input parameters and Monte Carlo random number streams. The total number of such runs can range from 10^2 to 10^5 and beyond. We detail relevant usage scenarios for this step in the next section.

4. **Visualization and analysis of results** – In the simplest case this might require 2-D plotting of one output parameter as a function of time. In the more typical case, some combination of 2-D plotting and 3-D imaging and/or animation is required to visualize the simulation's output.

At present, all simulation objects and run-time conditions are specified using a high-level Model Description Language (MDL) designed for readability by scientists. When a simulation is run, one or more MDL input files are parsed to create the simulation objects, and then execution begins for a specified number of time-step iterations. MCell is highly optimized for speed and for memory usage. This makes it possible to run individual simulations of complex structures on single processors rather than using parallel architectures.

So far, MCell simulations have been used to study *synapses*: structures used by nerve cells to communicate with themselves and other cells. MCell (and its predecessors) originally focused on the popular vertebrate neuromuscular junction, the synapse between a nerve cell and a muscle cell [49, 52]. MCell's Monte Carlo simulations have been successfully employed to obtain a variety of new results [5, 4, 54, 51, 52, 53, 52]. In addition, MCell has been in limited release [40, 39] to a worldwide group (~25) of Neuroscience and other research laboratories since 1997 [31, 48, 22, 21]. MCell is currently the object of many development efforts and current simulations are allowing scientists to explore new areas of cellular physiology.

2.2. MCell Usage Scenarios

Since MCell models are now approaching the level of structural and biochemical complexity present in living cells, the models typically contain numerous input parameters that can be varied independently. Consequently, simulations can span an enormous range of computational and data requirements. We detail here three relevant usage scenarios. We give orders of magnitude for the aggregate simulation CPU time assuming a single 2.0GHz Pentium processor.

(A) "Look & See" – A small number of MCell runs are used to determine the predicted behavior of the modeled system under limited input conditions – between 1 hour and several days of CPU time are required.

(B) Parameter Fitting – Tens to thousands of runs may be required to identify input parameter values which produce model output that matches given criteria such as experimental measurements – several weeks of CPU time.

(C) Parameter Sweep – The scale of individual simulations is similar to that for the parameter fitting scenario, but many thousands of runs are required to map a region of the input parameter space – anywhere from 1 month of CPU time to several years or decades.

Even though (A) has been the most common scenario in early stages of the MCell project, it is increasingly being replaced/complemented by scenarios (B) and (C). These last two scenarios require a tremendous amount of compute and storage resources. For (B), the user generally navigates toward a "best fit" by iterative parameter adjustments made according to some potentially ad-hoc heuristics. Thus a high degree of interactivity between the user and the computing resources is desirable to maximize productivity. Scenario (C) does not require interactivity as the user has already identified an "interesting" region of the parameter space to explore, perhaps via scenario (B). In [13], we gave an example of a small-scale simulation for (C), which required approximately 3 months of aggregate CPU time and and generated 94GB of raw output, which were then reduced to 600KB of synthesized output. Note that in all scenarios the CPU time of individual simulations can vary by several orders of magnitude solely depending on input parameter values.

2.3. Current Limitations

Currently, MCell imposes severe limitations on the usage scenarios described in the previous section. Ideally, users would have access to integrated software which guides them through the 4 steps identified in Section 2.1 and which enables all three usage scenarios on large-scale distributed computing environments.

In its current incarnation, MCell consists of a single executable which takes MDL files as input. The user is responsible for creating these files and for managing each MCell "project" in an ad-hoc fashion. The user is entirely responsible for running the individual simulations and collecting the output. This involves

labor-intensive activities such as resource selection, remote process creation/monitoring, fault-detection and restart, or application data movements. These tasks are generally performed via a set of ad-hoc scripts. In scenarios (B) and (C), this proves to be infeasible for most users given the desired scale of the simulations. In addition, there is no support for interactive simulation as required in scenario (B).

The MCell executable generally produces one or more output files. Users are responsible for averaging, post-processing, visualizing, and analyzing output files. In scenarios (B) or (C), this amounts to manipulating and mining large datasets, again in an ad-hoc fashion. MCell users typically employ the file system as a database for application data, which does not scale and cannot support scenarios (B) or (C) adequately. Generally, OpenDX [44] is used for most rendering and visualization tasks.

Our earlier work on the AppLeS Parameter Sweep Template (APST) [14] provides limited support for (C) in terms of running MCell simulation on Grid resources. It is a clear improvement over the traditional usage described above. However, since APST is general purpose, it fails to address most of the MCell-specific limitations listed above. As scenarios (B) and (C) are the future of MCell simulations for new scientific discoveries, it is critical to provide corresponding comprehensive software support. This is the overall goal of the VI project.

3. THE VIRTUAL INSTRUMENT

In this section we describe the VI project in terms of specific goals, relevant research issues, and software.

3.1. Goals

The main motivation behind the VI software development effort is to address the limitations highlighted in Section 2.3. More specifically, this is accomplished by providing the following capabilities:

Framework for MCell project development – The VI must provide a framework in which MCell users can easily specify individual "projects" in terms of input parameters, initial ranges for those parameters, number of repeats for Monte Carlo averaging, nature of output files, and nature of output post-processing steps (see Section 3.2 for a detailed description of MCell projects). The only component of an MCell simulation that cannot be automatically created is the core MDL code as it embodies the user's conceptual model. The VI must provide a framework for users to plug in their core MDL code and be freed of all other responsibilities. This framework can easily be embedded as part of a user interface. Finally, the VI should handle all application data management issues. This can be done, for instance, with a relational database.

User interface – At the moment, MCell does not provide any user interface to facilitate MCell project instantiation and management. The VI should provide a graphical interface for users to create MCell projects within the framework described above. In addition, that interface should be able to invoke data visualization and rendering capabilities provided by tools like OpenDX [44]. A full-fledged user interface for MCell is an intensive development project and is not our focus here. Instead, we aim at providing a simple interface that will allow us to explore computer science research issues involved when running large-scale distributed MCell simulations in scenarios (B) and (C).

Transparent deployment – The VI should handle the logistics of application deployment on behalf of the user. This includes resource discovery, authentication/authorization, remote job creation/control, application data movements, fault-detection and recovery. This can be achieved by building on the base Grid software infrastructure.

Interactive simulation – In order for scenario (B) to be effective, the VI must provide a way for users to interact with running simulations in order to guide, or *steer*, the computation. Users must be able to direct the search away from certain regions of the parameter space to be explored, and to concentrate on other regions, based on real-time intermediate application results. This requires that the VI allow the creation and cancellation of application tasks on the fly.

High performance – Given the scale of MCell simulations in scenarios (B) and (C), it is critical that the

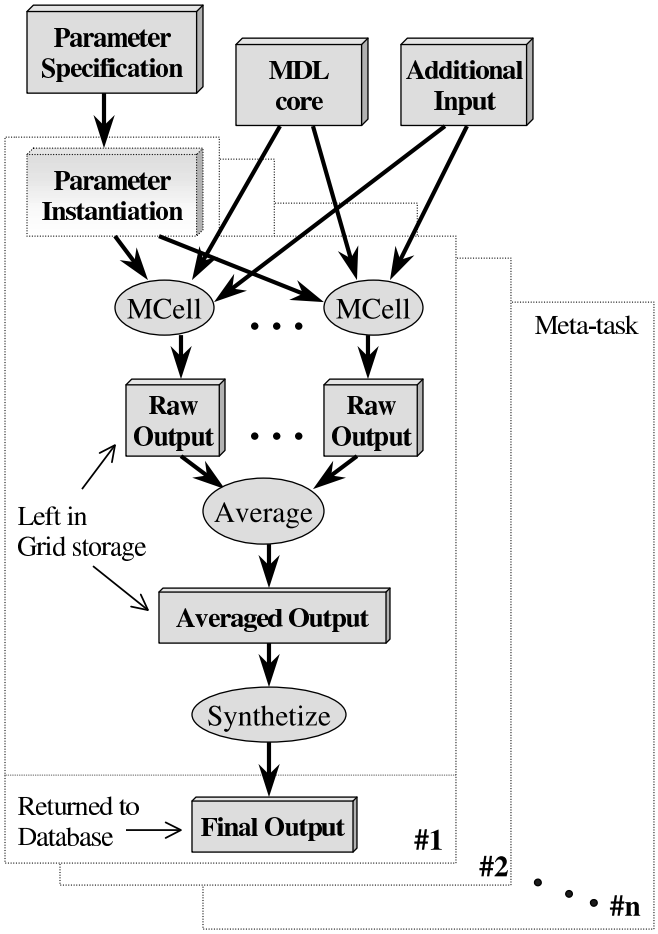

FIG. 1 The Structure of an MCell Project in the Virtual Instrument Framework: a project consists of n meta-tasks, and each meta-task consists of a number of identical MCell tasks whose outputs are averaged and synthesized into final output data.

VI exploit available resource effectively. This is to be achieved by the use of scheduling strategies, and can build on our previous work [16]. However, in this work, there is the added complexity of computational steering: how does one schedule (and re-schedule) an application whose computational goals are constantly being changed and/or refined by the user? Our goal is to develop resource allocation strategies that reduce application execution time (e.g. search time) in the presence of user steering.

3.2. MCell Projects in the VI Framework

Before describing our work on scheduling and on Grid software design and development, we introduce the notion of an *MCell project*, which can be created and executed by the VI user. The structure of a project is depicted in Figure 1 and consists of: (i) a set of parameter specifications (number of parameters, data types, initial value ranges); (ii) a set of MDL scripts written by the user – the MDL core; (iii) potential additional input files such as large geometry files that have been produced by 3-D reconstruction of electron microscope data. The MCell simulation consists of a (generally large) number of parameter space point evaluations, or *meta-tasks* (n meta-tasks are shown in the figure). Each evaluation consists of an instantiation of the parameter values and of a number of identical MCell tasks, each using different streams of random numbers

for Monte Carlo simulation. Each task produces raw output files that are then averaged and synthesized into final output. This final output is typically orders of magnitude smaller (in terms of bytes) than raw and averaged output. That final data must be analyzed by the user to understand the behaviors of the simulated biological system. This must be done on-the-fly to steer the computation in scenario (B). Note that raw and average output data is still of interest to the MCell user and may need to be retrieved in order to perform in-depth analyses. Note also that the amount of computation required for each meta-task depends on the parameter instantiations. In other words, some meta-task require more computation than some other (typically by up to three orders of magnitudes).

A run of an MCell project consists in executing large numbers of independent meta-tasks and generating both intermediate and final output.

3.3. Grid Computing Issues

The two main issues that we have explored in the VI project are that of Grid application development and application scheduling. We briefly discuss these issues below.

Resource Access and Data Management on the Grid – The VI project faces most of the issues inherent in Grid computing as it seeks to make the use of the Grid as transparent as possible so that the user can focus on the MCell simulation rather than on the logistics of application deployment. To that end, we reuse most of the available Grid infrastructure to achieve automatic resource discovery [19], resource access [20, 15, 6], security [26], distributed data management [18], and resource monitoring [62]. Our work on the VI builds on our experience when developing the APST project [14]. APST provides a simple, generic way to run parameter sweep applications and is currently used by MCell users for scenario (C).

One of the lessons we learned with APST is that targeting several underlying technologies for deploying user application makes it possible to (i) gain early acceptance from the users; (ii) increase the number of resources available to applications. This is the case because Grid computing is still an emerging technology and is not yet ubiquitous. Although the Open Grid Software Architecture (OGSA) standard [28] is rapidly gaining momentum, our goal is to enable MCell users to run simulations on their resources today.

Consequently, the VI targets a number of Grid services, which can be used simultaneously to expand the range of resources available to a single MCell simulation. In addition, the VI provides default mechanisms that use SSH to start remote jobs and move application data. SSH does not provide the levels of job control and the scalability offered by say, Globus [25]. However, our experience with APST is that users generally start using SSH mechanisms and progressively move towards Grid middleware technology as their simulation needs grow in scale. The main notion here is that current Grid application execution environments should be able to use whatever Grid middleware is available, but also degrade to default ubiquitous mechanisms if necessary. We expect this design to evolve to pure OGSA when the standard because more ubiquitous.

Given the life-span of MCell simulations, it is critical that the core VI software be resilient to software and hardware crashes. In addition the VI should automatically handle all application data management on behalf of the user. Consequently, we use a relational database in order to maintain persistent state about running MCell projects, data generated by those projects, and available resources. This database has two roles. First, it allows the VI software to be resilient to faults: all state is periodically saved into the database and can be used for restart. Second, it provides a structure for storing, retrieving, and mining application data, which is fundamental for achieving the first goal in Section 3.1. Our approach is to store only final application output data into the database (see Figure 1). Raw and intermediate output, which can be enormous, is left in place on remote Grid storage resources and can be downloaded on demand by the user.

Application Scheduling – A research direction explored in the VI project is that of application scheduling in a computational steering context. Scheduling sets of non-identical, independent tasks onto sets of distributed, heterogeneous resources has long been identified as an NP-hard problem [58]. Therefore, much research work has been dedicated to the development of appropriate scheduling heuristics (see [11] for a survey). Grid computing adds several challenges to the traditional scheduling problem: resources are not only heterogeneous, they exhibit dynamic performance behaviors due to sharing among users. Also, they are located on diverse network topologies interconnected over the wide-area. To address these issues, *adaptive scheduling* has been employed with success [9, 7]. As a result, we developed adaptive scheduling strategies

for MCell in our earlier work [16], focusing on scheduling data movement, data staging, and data duplication, with respect to storage and compute resource locations and characteristics.

In this work, we have to cope with the complexity added by computational steering, that is the problem of scheduling an application whose computational goals change over time according to potentially arbitrary user behaviors. Computational steering is a difficult problem that has been addressed by several researchers [45, 36, 60, 61, 30]. These efforts mostly addressed the problems of consistency of state among components of tightly coupled applications. In the limited context of MCell, consistency is not a key issue as the application consists of large sets of tasks which can be stopped and re-started independently, with little need for synchronization. Therefore, we focus on a resource allocation strategy that takes steering into account solely to achieve high performance.

The VI user typically employs the following general search strategy. The user's goal is to locate some particular point in the parameter space that satisfies some subjective criteria. An initial set of parameter space points (uniformly) distributed over the parameter space are selected for computation. As results come back from the VI, they are displayed to the user who can then assign *levels of importance* to regions of the parameter space. Regions with higher levels of importance are more promising and should therefore be completed sooner. This can be achieved by assigning appropriate fractions of the available compute resources to the exploration of each region. For instance, if the user has identified 3 regions that should be explored and assigned levels of importance 2, 2, and 1, then each of the first two regions should get 40% of the resources, and the last region should get 20%. The key idea is that rather than ordering regions by their level of importance and exploring the most promising regions first, it is more efficient to explore all regions concurrently but at different rates, to avoid being trapped in local optima in regions that were initially the most promising. Our contribution is that we reason at the resource allocation level rather than at the algorithmic level, which makes our approach applicable to a wide variety of steering behaviors and search algorithms (including both interactive user steering and search algorithms).

Our approach is to assign a priority to each point of the parameter space, corresponding to the level of importance of the region to which the point belongs. If each of the current n points being computed has a priority p_i, then point i should get $p_i / \sum_{j=1}^{n} p_j$ percent of the available resources. In [23] we have proposed and evaluated a number of strategies for computing priorities and for scheduling computational tasks according to these priorities. We refer the reader to that paper for all details. Our main result was that the use of priorities for allowing concurrent region explorations leads to significant performance improvements and we have quantified how priorities should be computed, given levels of importance that users assigned to regions.

3.4. Virtual Instrument Software

The Virtual Instrument software follows a strict object-oriented design and is constructed of three principal components: a software daemon to manage resources and remotely run jobs; a user interface to allow users to initiate, run, monitor, and stop MCell projects; and a database to store final application results and user-entered data. These components can run on separate machines.

Figure 2 depicts the interactions of the three main components of the VI architecture: the Daemon, the Interface, and the Database. The Daemon interacts with resources via Grid services. These services allow the Daemon to discover resources, start and control remote jobs, move data between distributed storage locations, and monitor resources as well as the running application. The Daemon uses the Database to store information such as the available resources, the user-defined specifications of running MCell projects, and the status of these running projects, including their pending tasks. To the greatest extent, the Daemon uses an out-of-core approach, so that if it fails, the relevant information about running MCell projects is in the Database. The only application data stored in the Database are MCell final output that can be visualized and analyzed by the user and used to steer further simulations. As depicted in Figure 2, the Database also keeps track of the location of all raw and intermediate data, which is left in place in Grid storage until it is explicitly retrieved by the user (see Section 3.3).

The Interface allows the user to steer the computation and to perform visualization. The Interface communicates control information to the Daemon, including commands to create, start, steer, and stop MCell projects or trigger the retrieval of a particular output file our final output data. Visualization of the data can be performed by OpenDX at the user's direction, as invoked from the Interface.

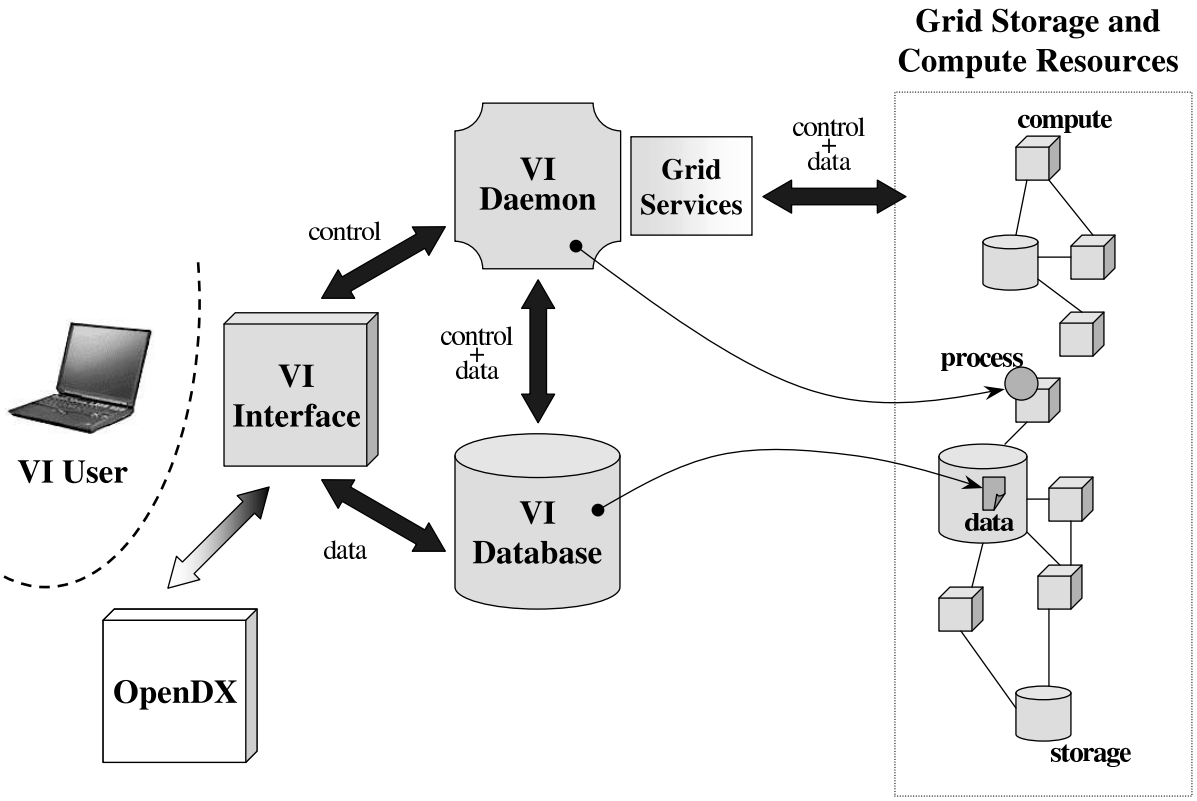

FIG. 2 The VI architecture: the three main components are the Daemon, the Database, and the Interface.

The main responsibility of the Daemon is to schedule and actuate file transfers and computations using available computational and network resources. These functions are performed by three classes within the Daemon: the Project class, the Scheduler class, and the Actuator class. The Project keeps track of all of the parameter space points and task inter-dependencies. For example, in Figure 1, it is the Project that is aware of the requirement to complete several runs of MCell with their parameter instantiations before running a post-processing task to average the output. The Scheduler retrieves information on tasks from the Project, sets their relative priorities (see Section 3.3), and assigns tasks to resources accordingly. The Scheduler is designed as a base class so that alternate scheduling strategies can be easily integrated as they are developed. After tasks have been assigned to resources by the Scheduler, the Actuator launches them on Grid resources. As with the Scheduler, the Actuator is designed as a base class, permitting specialization for various remote job execution and data transfer methods from various Grid middleware services.

The use of a relational database has several advantages. It makes the design of the Daemon more simple in terms of data structures, and makes it possible to recover from failures. In addition, the Interface does not need to implement an ad-hoc protocol with the Daemon, but can just pull data out of the Database in a standard fashion. In particular, it is possible to make complex SQL queries to mine application data. Furthermore, the use of a separate database allows users to start an MCell project, disconnect, and check the status of the simulation from any location.

3.5. Status of the Implementation

At the moment, the VI software consists of approximately 20,000 lines of C++, using the Standard Template Library and pthreads. We opted for MySQL to implement the Database as it is well accepted by the Linux community and provides a standard API. In the current release, the actuators within the Daemon target SSH, Globus's GRAM [20], NetSolve [15], and batch schedulers [1, 46] for starting/monitoring remote jobs, scp and GridFTP for moving application data on the Grid. Our implementation of the VI event system targets NWS [62] for resource monitoring. The VI Interface is still underway and at the moment we provide several interfaces. First, we have implemented a text-only interface for evaluation purposes. This interface allows us to gather information about user behaviors and requirements for converging towards a graphical interface. This interface is also written in C++ on top of VI components. We have also implemented a Web-based portal to the VI so that users can check on progress and perform simple data-mining tasks on the final application output. In addition, we have implemented a stand-alone Java-based GUI which is being used as the base for the final VI GUI. At the moment it implements the same functionality as the portal. Finally, we have implemented a simulator in order to evaluate our scheduling/steering strategies (see Section 3.3). The simulator is written with SimGrid [12, 38], a toolkit specially designed for the study of application scheduling in distributed computing systems, and has been integrated with the VI software. This allows us to simulate a variety of user behaviors and to test and validate the VI implementation throughout development.

An alpha version of the VI software was released to a limited number of MCell users/developers in February 2002 for evaluation. The software was subsequently enhanced and hardened and demonstrated at the SC'02 conference. The demonstration involved a simulation consisting of 11,360 MCell tasks and was executed on a testbed aggregating diverse computing resources in Japan and California (including batch-scheduled MPPs, batch-scheduled clusters, interactive clusters, and individual workstations). The simulation was launched and monitored by a user on the conference floor in Baltimore, Maryland. A beta version of the software was released to MCell users in February 2003.

The software, information about installation, and further details about the implementation can be found on the project's Webpage at `http://grail.sdsc.edu/projects/vi_itr`.

4. EXPERIMENTS

We have performed a large run of an MCell project on a Grid platform over the course of several days. Note that these experiments did not include any user steering as our goal here is solely to verify and evaluate the VI software design. The software supports steering and we refer the reader to [23] for a discussion of steering and scheduling.

4.1. The MCell simulation

In this section we give background for and describe the MCell simulation that we ran in our experiments. This simulation is called *r_disk* and targets a synapse. A synapse is a highly organized cellular structure that forms at the narrow junction between two neurons (or between a neuron and a muscle cell) in the nervous system. Chemical communication occurs across this synaptic cleft through a process called synaptic transmission. In synaptic transmission neurotransmitter molecules are released through the membrane of an excited presynaptic neuron and activate specific neurotransmitter receptor molecules on the membrane surface of a postsynaptic cell. Receptor activation results in a transient signaling event which might lead to excitation of the postsynaptic cell.

The *r_disk* MCell simulation attempts to map the parameter space of synaptic transmission for a canonical, simple model of synaptic geometry and assumes acetylcholine (ACh) and acetycholine receptor (AChR) as the neurotransmitter/receptor system. The synaptic geometry includes infinite planes for the pre- and postsynaptic membranes separated by a synaptic cleft distance fixed at 20 nm. There is a disk-shaped patch of receptors on the postsynaptic membrane. A fixed number of 10,000 ACh molecules are released from a point centered over the receptor patch. In this model we explore a 7 dimensional parameter space which includes the radius of the receptor patch, the diffusion constant of ACh, and 5 additional chemical kinetics parameters describing the various on-rates, off-rates, and conformation-change rates for the reaction mecha-

Site	CPUs	Access
TITECH	Presto-III cluster 20 dual-cpu Athlon, 1.59 GHz	Ssh
SDSC	Metor cluster 93 dual-cpu PIII, 730 MHz – 996 MHz	PBS
CSE	GRAIL workstations 12 Athlon and PIII, 200MHz – 1.5GHz	4 via Ssh, 8 via Globus

TABLE 1
Testbed for the experiments presented in Section 4.

nism between ACh and AChR. The overall computation time for this simulation amounts to approximately 329 days of computation on a 1.5GHz Pentium III.

4.2. The testbed

The testbed for our experiments consists of resources at three sites: the Tokyo Institute of Technology (TITECH) in Japan, the San Diego Supercomputer Center (SDSC) and the Dept. of Computer Science and Engineering (CSE), both at the University of California, San Diego (UCSD). All resources run some version of the Linux operating system. Table 1 summarizes overall characteristics of the resources and how they are accessed. Two of the sites, TITECH and CSE, provide interactive resources. Resources at TITECH can be reserved via an e-mail reservation system, and thus were not time shared with other applications for our run. The resources at CSE were not reserved and were thus time-shared with local user jobs. The resources at SDSC are accessed via the Portable Batch System [46] batch queuing system. In these runs, the VI software used 4 simultaneous submissions to the PBS queue, with requests for 4, 8, 16, and 32 processors. The Daemon and Database, as well as the user, are located on one of the CSE machines. At the end of the execution all synthesized application output is available in the Database and all intermediate application output is accessible in remote storage.

4.3. Results

Figure 3 depicts the cumulative amount of computation performed with respect to time. We show a curve for the total platform (`Total`) as well as a breakdown per site (`SDSC`, `TITECH`, and `CSE`). Note that we do not plot the cumulative number of tasks completed. Indeed, this number is misleading as MCell tasks vary widely in computational costs. For this simulation, the amount of computation required varied by as much as a factor 200 between MCell tasks. Instead, we ran an off-line benchmark of the MCell tasks involved in this simulation on reference CPUs (identical nodes of the Presto-III cluster). This allowed us to associate each task in the simulation with a "relative computational cost" (i.e. the amount of time the task requires on the reference CPU). This provides the basis for the cumulative computational cost plotted on the y-axis of Figure 3. An alternate approach would have been to use a performance model, but no accurate performance model for MCell tasks was available at the time of these experiments. In addition, the computational cost estimates obtained via the benchmark were used to estimate wallclock time requests passed to the PBS batch system on the Meteor cluster.

We make three main observations on the data presented in Figure 3:

(i) All sites contribute to the computation. As expected the site contributing the most to the computation is SDSC. Overall, SDSC contributed to 83% of the overall computation, TITECH to 13%, and CSE to 4%. The entire simulation (which amounts to approximately 329 days of computation on a 1.5GHz Pentium III) was completed in a little over 150 hours.

(ii) All curves level off before the end of the execution, due to the typical "waiting for the last task" syndrome. When scheduling a set of independent tasks, it is generally a good idea to schedule long tasks first [35, 34]. However, as discussed earlier, we do not have a performance model for MCell tasks.

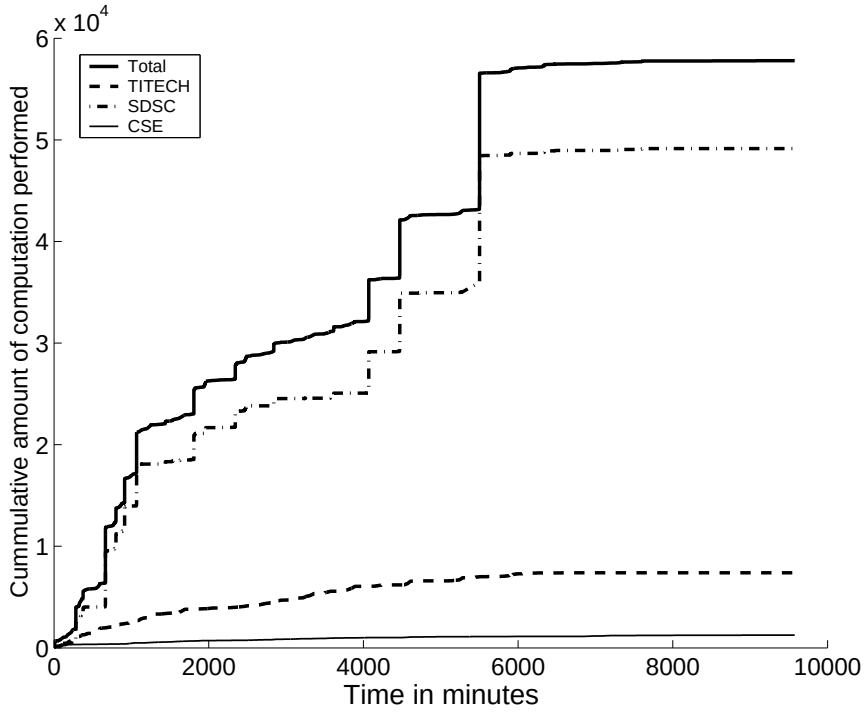

FIG. 3 Cumulative amount of computation performed during the experimental run on all sites and per site.

In other words, there is no way for the VI scheduler to determine automatically which tasks will be long (or short). As a result, it is likely that a number of long tasks will be scheduled at the end of the execution, leading to the leveling of the curves in Figure 3. A possible approach would be to run the aforementioned off-line benchmark and have the scheduler use the benchmark timings to make scheduling decisions. However, this is time and resource consuming, and it would need to be done for each new MCell project. But more importantly, note that in a real usage scenario the user would continually steer the simulation and add new tasks. Therefore, reducing the execution time of a fixed number of tasks is not the focus of this work (unlike that of our previous work [16, 17]). In this work we focus on overall throughput and thus do not consider the "waiting for the last task" problem to be significant.

(iii) The curve for SDSC exhibits more of a step pattern than the curve for TITECH or CSE. This is due to the use of the PBS batch system at SDSC: MCell tasks are submitted in batches, each batch spends time waiting in the queue, and thus tasks in batches tend to return results in bursts. By contrast, the TITECH resources are available in interactive mode and thus complete tasks at a relatively steady rate. This raises an interesting question regarding the use of batch-scheduled resources for running applications that consist of many independent, small tasks, as well as for running interactive applications: how many simultaneous requests and which request sizes should be sent to a batch-scheduled resource? In the current VI software these values can be fixed arbitrarily (as seen in Section 4.2). We leave the investigation of how these values should be chosen as maximize throughput as future work.

Figure 4 plots the cumulative number of failures that occurred during the entire execution for the SDSC and CSE resources. We do not show a curve for the TITECH site as no failure was experienced during this experiment on TITECH resources. 60 failures occurred during the run, meaning that on average the VI experienced approximately 0.6 failures per hour. These failures were caused by several factors, including actual resource downtimes, network time-outs, and software failures (e.g. SSH disconnections). The VI software detects failures, attempts a number of retries, and then marks resources as "failed" for a fixed amount of time before attempting other retries. Therefore some of the failures plotted in Figure 4 correspond to several occurrences of a failure for a single resource.

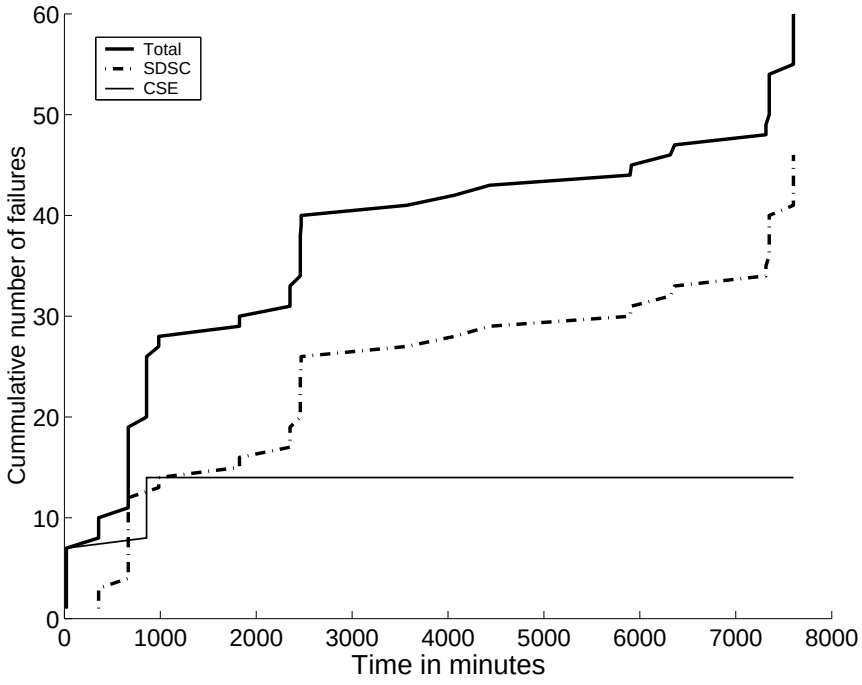

FIG. 4 Cumulative number of failures during the experimental run for SDSC and CSE.

4.4. Discussion

The experimental results obtained with the first VI implementation provide convincing evidence that:

(i) The VI software is functional and makes it possible to support large MCell runs on large-scale Grid testbeds. Furthermore, the software shields the user from the logistics of application deployment and provides the illusion of a desktop execution.

(ii) The VI software design is effective. The use of an out-of-core relational database for storing all information pertaining to the resources and the application proved to be scalable. In these experiments and others that are not reported in this paper, the use of the database was never a bottleneck and scales with the size of the application and the platform. In fact, the main bottleneck for application deployment is the overhead for launching remote computations on distant Grid resources.

(iii) The VI implementation is resilient to resource faults and failures. Our experiments show that failures occur in Grid environments at a non-negligible rate and due to several causes. The VI software appropriately detects failures and attempts a number of retries. This is critical for long-running applications such as MCell.

While a number of features and capabilities can be added to the VI software, in particular to the user interface, it provides sufficient functionality and robustness to enable the new generation of MCell simulations. Furthermore, all logistics of application deployment are hidden to the user and application data is automatically managed on behalf of the user. We conclude that the current version of the software meets the goals outlined in Section 3.1.

5. RELATED WORK

Our work is related to a number of large efforts that seek to provide Grid application execution environments for scientific simulations [33, 42, 47]. Related works also include portal activities [43, 57] and the VI software could ultimately be integrated as a user portal.

Computational Steering has been an active field of research and several projects have provided models, methodologies, and software for steering scientific applications (SCIRUN [45], VASE [36], Progress [60], Magellan [61], CUMULVS [30]). One of the main challenges addressed in these works is the notion of state consistency. Several techniques from the area of distributed systems and fault-tolerance have been used successfully to build high performance consistent computational steering environments. Our work is related to those efforts in that we provide computational steering capabilities. However, given the structure of MCell simulations, i.e. parallel searches with loose task and data synchronization requirements, state consistency is not a crucial issue. Therefore, our work focuses mostly on performance issues and proposes a scheduling/steering strategy based on task priorities for appropriate resource sharing.

This work builds on our earlier work on the AppLeS Parameter Sweep Template (APST) [14], which is related to projects such as Nimrod [2] or ILAB [63]. APST provides a generic Grid application execution environment for Parameter Sweep Applications. These applications consist of large numbers of computational tasks that exhibit few or no interdependencies. This category of applications encompasses many methodologies such as Monte Carlo simulations, parametric studies, and parameter searches, and arises in many fields of science and engineering. This work uses APST as a learning experience to provide a full-fledged execution environment customized for MCell. APST addresses a few of the limitations listed in Section 2.3 and is currently used for medium-scale MCell parameter sweep runs. Neither APST, Nimrod, nor ILAB provide capabilities for computational steering.

6. FUTURE WORK AND CONCLUSIONS

In this paper we have presented the Virtual Instrument (VI) project, which targets the deployment of large-scale, interactive MCell simulations. MCell is a molecular biology simulator, which has gained great popularity in the computational neuroscience community. Although the current MCell software provides basic capabilities to run simulations, it does not enable interactive simulation, and leaves many responsibilities to the user in terms of deployment, scheduling, and data management. These limitations preclude the use of MCell for large-scale executions, especially on the Grid platform. The goal of the VI project is to provide an integrated Grid execution environment for MCell that offers interactive computational steering capabilities. We have described contributions of our Grid software development effort and have given a brief account of our work in the area of application scheduling. We have then described the VI software in detail and presented validation experiments conducted for a real-world MCell application on a multi-site Grid testbed. These results are conclusive as they show that the VI software is functional and the design scalable.

Many future directions are currently being explored in this project. A beta version of the VI software was recently released to MCell users. Our ultimate goal is to deploy the software in a production environment to (i) further validate our implementation; (ii) log information about usage and learn about user behaviors; (iii) enable new disciplinary results. Ultimately, the Virtual Instrument will have a large and quantifiable impact on the MCell community. It will lead to new scientific advances that would not be possible without the Grid platform and without our fully integrated software environment.

ACKNOWLEDGEMENT

The authors wish to acknowledge the San Diego Supercomputer Center and the Matsuoka laboratory at the Tokyo Institute of Technology for providing computational resources that made this work possible.

REFERENCES

[1] IBM LoadLeveler User's Guide, 1993. IBM Corporation.

[2] J. Abramson, D. Giddy and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), Cancun, Mexico*, pages 520–528, May 2000.

[3] G. Allen, J. Shalf, W. Benger, T. Dramlitsch, T. Goodale, H. H.-C., G. Lanfermann, A. Merzky, T. Radke, and E. Seidel. Cactus Tools for Grid Applications. *Cluster Computing*, 4(3), July 2001.

[4] L. Anglister, J. R. Stiles, and M. M. Salpeter. Acetylcholinesterase density and turnover number at frog neuromuscular junctions, with modeling of their role in synaptic function. *Neuron*, 12:783–794, 1994.

[5] T. M. Bartol, B. R. Land, E. E. Salpeter, and M. M. Salpeter. Monte Carlo simulation of miniature endplate current generation in the vertebrate neuromuscular junction. *Biophys. J.*, 59(6):1290–1307, 1991.

[6] J. Basney and M. Livny. Deploying a High Throughput Computing Cluster. In *High Performance Cluster Computing*, volume 1, chapter 5. Prentice Hall, May 1999.

[7] F. Berman. *The Grid, Blueprint for a New computing Infrastructure*, chapter 12. Morgan Kaufmann Publishers, Inc., 1998. Edited by Ian Foster and Carl Kesselman.

[8] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, L. J. Dennis Gannon, K. Kennedy, C. Kesselman, D. Reed, L. Torczon, and R. Wolski. The GrADS project: Software support for high-level grid application development. *International Journal of High Performance Computing Applications*, 15(4):327–344, Winter 2001.

[9] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-Level Scheduling on Distributed Heterogeneous Networks. In *Proc. of SC'96, Pittsburgh*, 1996.

[10] M. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed Processing of Very Large Datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, October 2001.

[11] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems. In *Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99)*, pages 15–29, Apr. 1999.

[12] H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pages 430–437, May 2001.

[13] H. Casanova, T. Bartol, J. Stiles, and F. Berman. Distributing MCell Simulations on the Grid. *The International Journal of High Performance Computing Applications*, 14(3):243–257, 2001.

[14] H. Casanova and F. Berman. *Parameter Sweeps on the Grid with APST*, chapter 33. Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Publisher, Inc., 2003.

[15] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[16] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, May 2000.

[17] H. Casanova, G. Obertelli, H. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the Grid. In *Proceedings of SC'00*, November 2000.

[18] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 2000. to appear.

[19] C. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE Symposium on High-Performance Distributed Computing*, August 2001.

[20] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[21] D. Egelman, R. King, and P. Montague. Interaction of nitric oxide and external calcium fluctuations: a possible mechanism for rapid information retrieval. *Progress in Brain Research*, 118:199–211, 1998.

[22] D. Egelman and P. Montague. Computational properties of peri-dendritic calcium fluctuations. *J. Neurosci.*, 18(21):8580–8589, 1998.

[23] M. Faerman, A. Birnbaum, H. Casanova, and F. Berman. Resource Allocation for Steerable Parallel Parameter Searches. In *Proceedings of the Grid Computing Workshop*, pages 157–169, Nov. 2002.

[24] I. Foster and C. Kesselman, editors. *The Grid, Blueprint for a New computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1998.

[25] I. Foster and C. Kesselman. *Globus: A Toolkit-Based Grid Architecture*, pages 259–278. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.

[26] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[27] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3), 2001.

[28] I. Foster, J. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, June 2002. Open Grid Service Infrastructure WG, Global Grid Forum.

[29] The GAMESS Portal. `https://gridport.npaci.edu/GAMESS/`.

[30] G. Geist, J. Kohl, and P. Papadopoulos. CUMULVS: Providing Fault Tolerance, Visualization, and Steering of Parallel Applications. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):224–235, 1997.

[31] J. Gieger, A. Roth, B. Taskin, and P. Jonas. Glutamate-mediated synaptic excitation of cortical interneruons. In P. Jonas and H. Moyner, editors, *Handbook of Experimental Pharmacology, Retinoids, Ionotropic glutamate receptors in the CNS*, volume 141, pages 363–398, Berlin, 1999. Springer-Verlag.
[32] The Grid Portal Collaboration. http://www.ipg.nasa.gov/portals/.
[33] GriPhyN Webpage. `http://www.griphyn.org`.
[34] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.
[35] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein. Load-sharing in heterogeneous systems via weighted factoring. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 318–328, Jun 1996.
[36] D. Jablonowski, J. Bruner, B. Bliss, and R. Haber. VASE: The Visualization and Application Steering Environment. In *Proceedings of Supercomputing 1993*, pages 560–569, 1993.
[37] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 2003. to appear.
[38] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan*, May 2003. to appear.
[39] MCell Webpage at the Pittsburgh Supercomputer Center. `http://www.mcell.psc.edu`.
[40] MCell Webpage at the Salk Institute. `http://www.mcell.cnl.salk.edu`.
[41] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. An Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *Proceedings of the Grid Computing Workshop, Baltimore*, pages 274–279, November 2002.
[42] National Virtual Collaboratory for Earthquake Engineering Research Webpage. `http://www.neesgrid.org`.
[43] J. Novotny. The Grid Portal Development Kit. *Concurrentcy and Computation: Practice and Experience, Special Issue on Grid Computing Environments*, 2002. May.
[44] OpenDX Webpage. `http://www.opendx.org`.
[45] S. Parker, M. Miller, C. Hansen, and C. Johnson. An integrated problem solving environment: The SCIRun computational steering system. In *Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS-31), vol. VII*, pages 147–156, January 1998.
[46] The portable batch system. `http://www.openpbs.com`.
[47] Particle Physics Data Grid Webpage. `http://www.ppdg.net`.
[48] R. Rao-Mirotznik, G. Buchsbaum, and P. Sterling. Transmitter concentration at a three-dimensional synapse. *J. Neurophysiol.*, 80(6):3163–3172, 1998.
[49] M. M. Salpeter. *The Vertebrate Neuromuscular Junction*, pages 1–54. Alan R. Liss, Inc., New York, 1987. Edited by Salpeter, M. M.
[50] J. R. Stiles and T. M. Bartol. Monte Carlo methods for simulating realistic synaptic microphysiology using MCell. In E. DeSchutter, editor, *Computational Neuroscience: Realistic Modeling for Experimentalists*, Boca Raton, 2001, in press. CRC Press.
[51] J. R. Stiles, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. Monte Carlo simulation of neurotransmitter release using MCell, a general simulator of cellular physiological processes. In J. M. Bower, editor, *Computational Neuroscience*, pages 279–284, New York, NY, 1998. Plenum Press.
[52] J. R. Stiles, T. M. Bartol, M. M. Salpeter, E. E. Salpeter, and T. J. Sejnowski. Synaptic variability: new insights from reconstructions and Monte Carlo simulations with MCell. In W. M. Cowan, T. C. Südhof, and C. F. Stevens, editors, *Synapses*, pages 681–731, Baltimore, 2001. Johns Hopkins University Press.
[53] J. R. Stiles, I. V. Kovyazina, E. E. Salpeter, and M. M. Salpeter. The temperature sensitivity of miniature endplate currents is mostly governed by channel gating: evidence from optimized recordings and Monte Carlo simulations. *Biophys. J.*, 77:1177–1187, 1999.
[54] J. R. Stiles, D. Van Helden, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. Miniature endplate current rise times <100 μs from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. *Proc. Natl. Acad. Sci. U.S.A.*, 93:5747–5752, 1996.
[55] T. Suzumura, H. Nakada, M. Saito, S. Matsuoka, Y. Tanaka, and S. Sekiguchi. The Ninf Portal: An Automatic Generation Tool for the Grid Portals. In *Proceedings of Java Grande*, pages 1–7, November 2002.
[56] The Telescience Portal. `https://gridport.npaci.edu/Telescience/`.
[57] M. Thomas, S. Mock, J. Boisseau, M. Dahan, K. Mueller, and S. Sutton. The GridPort Toolkit Architecture for Building Grid Portals. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, August 2001.
[58] J. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:434–439, 1975.
[59] R. van Nieuwpoort, T. Kielmann, and H. Bal. Efficient Load Balancing for Wide-area Divide-and-Conquer Applications. In *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'01)*, pages 34–43, June 2001.

[60] J. Vetter and K. Schwan. PROGRESS: A Toolkit for Interactive Program Steering. In *Proceedings of the 1995 International Conference on Parallel Processing*, pages 139–149, 1995.

[61] J. Vetter and K. Schwan. High Performance Computational Steering of Physical Simulations. In *Proceedings of IPPS'97*, pages 128–132, 1997.

[62] R. Wolski. Dynamically Forecasting Network Performance Using the Network Weather Service. In *6th High-Performance Distributed Computing Conference*, pages 316–325, August 1997.

[63] M. Yarrow, K. McCann, R. Biswas, and R. Van der Wijngaart. An Advanced User Interface Approach for Complex Parameter Study Process Specification on the Information Power Grid. In *GRID 2000, Bangalore, India*, December 2000.