The Virtual Instrument: Support for Grid-enabled Scientific 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 Gockay 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 distributed, heterogeneous resources, or *Computational Grids*, have emerged as popular platforms for large-scale scientific applications. This paper presents the *Virtual Instrument* project which targets those platforms. More specifically, the project seeks to provide an integrated application execution environment that enables end-users to run and interact with running scientific simulations on the Grid. This work is performed in the specific context of a computational biology application: MCell. Even though MCell provides the basis for running simulations, its capabilities are currently limited in terms of scale, ease-of-use, and interactivity. Those 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 need for an MCell Virtual Instrument. We then introduce a scheduling strategy that exploits the structure of MCell simulations and uses task priorities to accommodates computational steering. Finally, we describe our innovations and contributions in terms of Grid software design and development.

Key Words: Grid Computing, Computational Neuroscience, Interactive Simulation, Computational Steering, Scheduling, Event System.

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

1. INTRODUCTION

Computational Grids [21, 24] are large collections of resources (computational devices, networks, on-line instruments, storage archives, etc.) that have enormous potential and are becoming popular platforms for running large-scale, resource-intensive applications. Many challenges are to be addressed in order to provide the necessary mechanisms for accessing, discovering, monitoring, and aggregating Grid resources. Consequently, a tremendous effort has been made and is still underway to provide a number of middleware technologies [28, 29, 22, 32, 48]. However, even though middleware provides fundamental building blocks, it is not designed to be used directly by Grid users. Instead, Grid application execution environments must be provided with the goal of hiding the Grid complexity, both in terms of hardware and software, from the end-user. One approach is to build "Grid portals" which provide a familiar Web browser interface to Grid services [31, 30]. Alternatively, several projects implement programming models that provide abstractions for building Grid applications [13, 41]. Finally, another possibility is to build integrated software environments that are either general purpose [8], or targeted to specific domains and applications. Our work belongs in that 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.

This work is performed in the specific context of the MCell application [39, 38, 55, 52]. MCell is a computational biology simulation framework that is currently used by neuroscientists to study diffusion and chemical reactions of molecules in living organisms. A single execution instance of MCell consists of multiple computational simulations, each producing output that is then analyzed by neuroscientists *en masse*. Even though MCell provides the basis 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 those phenomena while executing, a great deal of CPU time could be saved. This and other limitations preclude usage scenarios that are critical for scientific advances. The goal of the VI is to alleviate most of the current limitations in 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 highlight two main contributions of our work. First we introduce a scheduling strategy that uses task priorities to accommodate computational steering. We present simulation results obtained to validate that strategy. Second, we describe accomplishments in terms of Grid software design and development. This development primarily entails the realization of an event system capable of delivering user and resource events to the scheduler, and of an overall architecture for generalized VI Grid software.

We describe our efforts 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, present our contributions in terms of scheduling and Grid software, and describe the VI's software design and implementation. Section 4 discusses related work and Section 5 concludes the paper with future research and development directions.

2. MOLECULAR BIOLOGY SIMULATIONS WITH MCELL

2.1. MCell Overview

MCell [39, 38, 55, 52] 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 [53]. 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 [51].

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 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 [50, 53]. MCell's Monte Carlo simulations have been successfully employed to obtain a variety of new results [5, 2, 55, 52, 53, 54, 53]. In addition, MCell has been in limited release [39, 38] to a worldwide group (~25) of Neuroscience and other research laboratories since 1997 [27, 49, 20, 19]. 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 magnitudes for the aggregate simulation CPU time assuming a single workstation (the fastest currently available on the market).

(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 [12], we gave an example of a small-scale simulation for (C), which required approximatively 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 those 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. Those 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 by 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 use the file system as a database for application data, which does not scale and cannot support scenarios (B) or (C). Generally, OpenDX [44] is used for most rendering and visualization tasks.

Our earlier work on the AppLeS Parameter Sweep Template (APST) [15, 4] provides limited support for (C) in terms of running MCell simulation on Grid resources. It is a clear improvement over the traditional usage. However, since APST is completely generic, 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 one of the goals 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. However, all the aforementioned components can be standardized. The VI must then 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 must handle all application data management issues. This can be done, for instance, with a relational database.

User interface – At the moment, MCell lacks a user interface. 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]. Clearly, a full-fledged user interface for MCell is a multi-year 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. Our goal is also to allow current MCell users to perform first generation runs under scenarios (B) and (C).

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

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. The question of computational steering has been explored in other work and poses many challenges, including precise application control, high performance, and consistency of state (see Section 4 for a discussion of related work in that area).

High performance – Given the scale of MCell simulations in scenarios (B) and (C), it is critical that the VI exploit available resource effectively. This is to be achieved by the use of scheduling strategies, and can build on our previous work [14]. 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

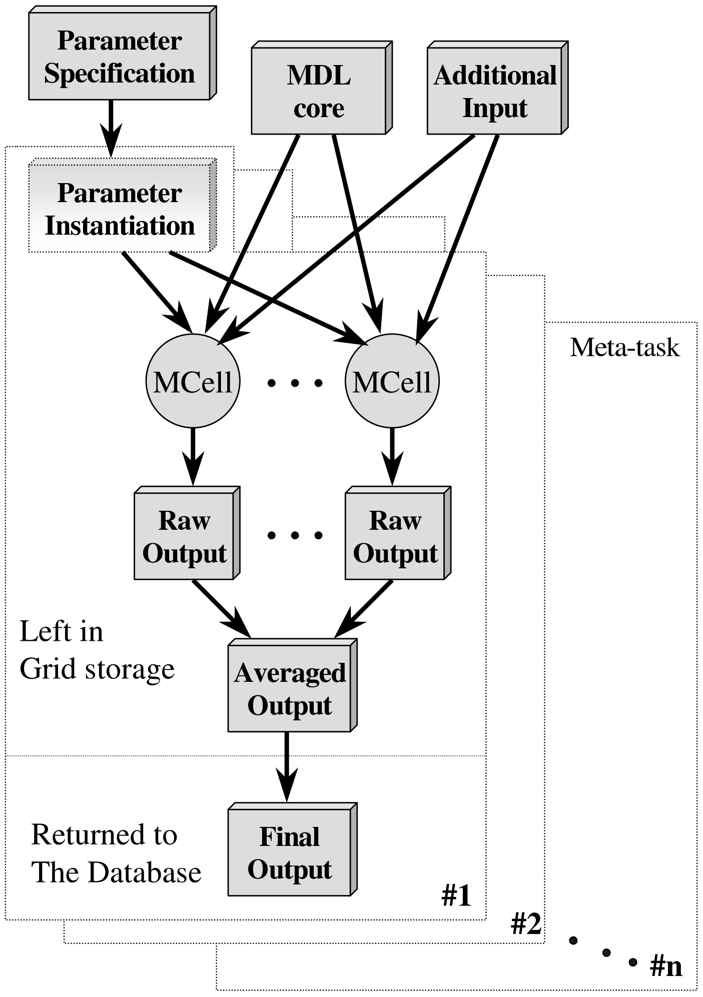

FIG. 1 The Structure of an MCell Project in the Virtual Instrument Framework: a project consists of n meta-tasks, and each meta-task consists of a number of identical MCell tasks whose outputs are averaged and synthesized into final output data.

being changed and/or refined by the user?

3.2. MCell Project in the VI Framework

Before describing our work on scheduling and on Grid software design and development, we formalize 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 on 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 the Monte Carlo simulation. Each task produces raw output files that are then averaged and synthesized into final output. That 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 in order 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.

A run of an MCell project then consists in executing large numbers of independent meta-tasks and retrieving both intermediate and final output. In the following two sections we present research issues and our initial work for scheduling/steering and for targeting the VI to the Grid.

3.3. Scheduling/Steering Issues

Scheduling sets of independent tasks onto sets of distributed resources has long been identified as an NP-hard problem [57]. Therefore, much research work has been dedicated to the development of appropriate scheduling heuristics (see [10] 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 those issues, *adaptive scheduling* has been employed with success [9, 7]. As a result, we developed adaptive scheduling strategies for MCell in our earlier work [14]. That work focused 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, 35, 58, 59, 25]. Those 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 and without need for synchronization. Therefore, we focus on designing a scheduling strategy that take steering into account in order to achieve high performance.

As illustrated in Figure 1, MCell simulations consist of many independent meta-tasks, each corresponding to the evaluation of one point in a large multi-dimensional parameter space. In this work we currently ignore issues concerning application input/output data (those issues were explored in [14]). We assume that the VI user employs the following search strategy. The user's goal is to locate some particular point in the parameter space that satisfies some subjective criteria. She initially selects parameter space points uniformly distributed over the parameter space. 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 the first two regions should both get 40% of the resources, and the last region should get 20%. 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. Region exploration is performed by refining the resolution within that region, i.e. by computing increasingly dense parameter space point meshes. There are several ways to increase mesh density in regions, e.g. increasing the resolution by some factor in all dimensions of the parameter space (progressive refinement), or by selecting a number of random points in the region (random local search).

Development of scheduling/steering strategies for the VI faces an inherent difficulty: user behaviors are unknown until the VI software is in production mode. In addition, MCell users will use the VI to search for data patterns that are rather subjective and therefore difficult to specify a-priori (e.g. look for something that is "interesting"). We make the observation that *user-directed searches share commonalities with algorithmic searches*: typical user steering patterns are really instances of search algorithms that use a combination of

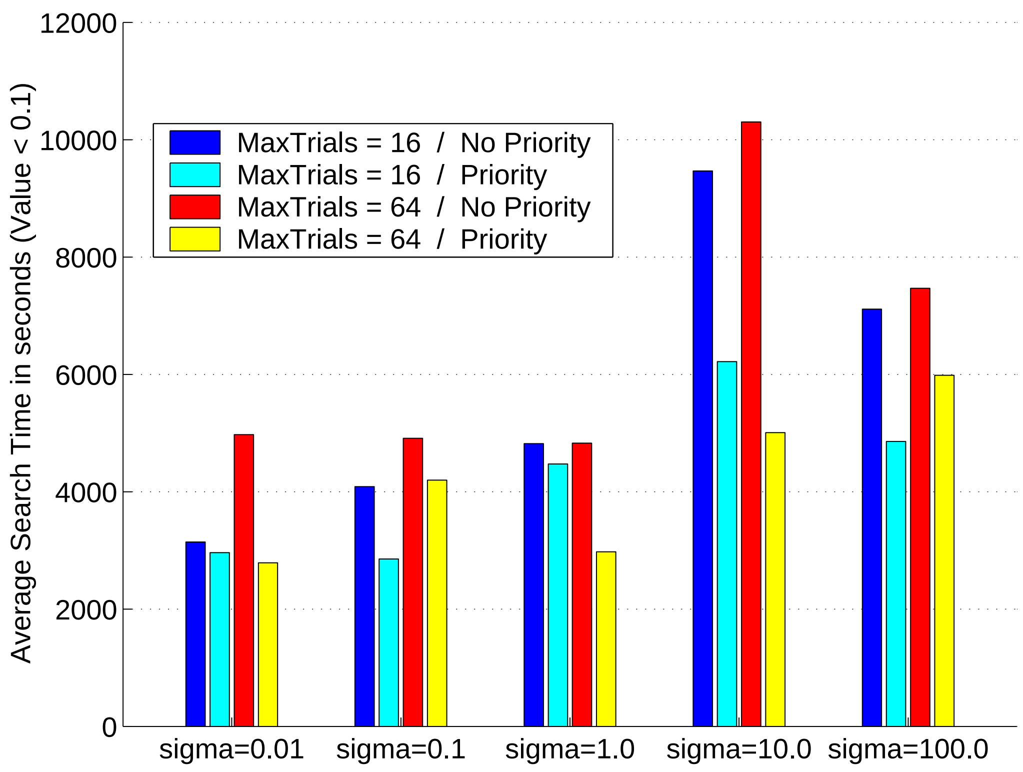

FIG. 2 Legend

local and global searches. Rather than waiting for a fully developed VI software which is used in production mode by neuroscientists, we drive our scheduling research by emulating user behavior: our approach is to use global/local optimization algorithms to optimize several well-known objective functions. Remember that our goal is to evaluate the effectiveness of priority-based scheduling to reduce overall search time. The behavior of an actual MCell user is arguably different from that of an automated algorithm but we believe that the similarities are sufficient for us to evaluate our scheduling approach. Also, search algorithms might be provided as a VI feature when users can clearly specify their goal (e.g. look for the minimum of this function). Finally, this approach allows us to make a broader contribution as we validate that our scheduling strategy is applicable to parallel search algorithms.

We implemented a simulator using the Simgrid toolkit [11] in order to study a wide range of scenarios in terms of search algorithms and objective functions. Figure 2 shows one set of results obtained with that simulator. We simulated a search for a global minimum over the well-known positive "hard-to-optimize" 3-D Griewank function [56]. That function is parametrized by a positive real number, σ. For five values of σ we simulated a popular global/local search pattern. The search starts with a *generation* of random parameter space points at which the Griewank function is evaluated. Then, at each generation, local searches are conducted around points from the previous generation. For each local search, the best 2 points (i.e. the ones with the lowest Griewank value) are propagated to the next generation. Each local search consists of a maximum number of trials, that is: if a local search does not return a better result than its parent then another local search is attempted until the maximum number of such trials is reached. We use two values for $maxTrials$: 16 and 64. The first one consists of a rather fast aggressive search, where the second one attempts to be more thorough by doing a larger number of local searches before giving up on one region.

We claim that this search algorithm is representative of an execution of the VI where the user would steer the simulation at each generation by selecting promising regions to explore. The simulated platform considered here consists of a single, "continuously partitionable" processor. In other words, it is possible to assign any fraction of that processor's compute power to a given task. Therefore, our assumption is that it is possible to implement task priorities exactly. For each steering pattern (MaxTrials=16 and MaxTrials=64), we simulated execution with and without task priorities. When used, priorities are simply computed as the inverse of the Griewank values (close to zero implies a promising region, hence high priority). The results on Figure 2 show average search times (over 100 repetitions), where we declare a search complete when the Griewank value is under 0.1. In those conditions the search generally used under 20 generations.

As expected, results vary with σ and the search patterns. It is difficult to precisely understand which search pattern is more effective in which situations. Indeed, much research work has been dedicated to understanding, designing, and tuning search algorithms. However, our focus here is on scheduling. We can see that, in all cases, the use of priorities reduces the overall search time, by 5% to 34% for $maxTrial = 16$, and by 14% to 51% for $maxTrial = 64$. Our claim is that for many steering patterns when using the VI, the use of priorities when scheduling tasks onto the Grid will reduce search time. This is further substantiated by many other results that we obtained with many objective functions (both continuous and discrete) for many search algorithms.

These results are clearly encouraging. However, note that in these simulations we assume that tasks get exactly the portion of the resources dictated by their priorities. In practice, it is not possible to assign precise quanta of Grid resources to tasks of an application. This is due in part to the fact that Grid resources are heterogeneous and with dynamic performance behavior. Also, the application, Grid middleware, and/or operating systems might not provide the necessary degree of control. An actual implementation of a scheduling strategy can only hope to achieve an approximation of the priorities. For instance, the VI scheduler could decide to assign tasks to resources in order of priorities and let each task run to completion uninterrupted. Alternatively, using job control mechanisms, the VI could interrupt and checkpoint tasks in favor of others, thereby achieving some level of time-sharing among tasks. This would lead to a better approximation of the original priorities at the added cost of checkpointing overhead. We are currently conducting a quantitative study of those trade-offs.

We have given here broad directions and early results for scheduling/steering in the context of the VI project. We will report on the details of our work on scheduling in upcoming papers.

3.4. Grid Computing Issues

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 needed so that the user can focus on the MCell simulations rather than on the logistics of application deployment. To that end, we reuse many of the recent Grid middleware efforts to achieve automatic resource discovery [17], resource access [18, 13, 6], security [23], distributed data management [16], and resource monitoring [61].

In this section we describe specific issues in which the VI approach makes a contribution to Grid computing. Some of these contributions come from our experience with the APST project [15, 4]. APST provides a simple, generic way to run parameter sweep applications and is currently used by MCell users for scenario (C). When designing the VI architecture we were able to learn from and improve on APST's principles.

3.4.1. Event System

Efficient scheduling for steerable applications on the Grid relies on three sources of information: application resource requirements, user steering input, and available resource performance. These three information sources can be characterized by measures of their *uncertainty* and their *dynamism*. Both measures are necessary for automatic schedulers to make effective decisions in Grid settings. The most natural and scalable architecture for delivering such information to the application scheduler is via a *distributed event system*. Rather that having the scheduler actively poll a potentially large number of resources, an event system monitors performance conditions on behalf of the scheduler, and then notifies it when a scheduler-specified set of conditions occurs. For example, the scheduler might determine that a particular machine offers excellent execution performance for certain application tasks, but that (due to the current load on the machine) insufficient memory is available to support those tasks. It is infeasible to have the scheduler constantly poll the resource to determine if and when enough memory becomes free. While it might work for a small number of machines, the scheduler would spend all of its time polling (potentially receiving negative answers most of the time) if a large number of resources and characteristics are to be considered. Therefore, a distributed event system is one of the necessary components for software such as the VI.

Key to the construction of the event system is an *event model* – a set of abstractions and the rules for their interaction which, when implemented, will provide efficient and useful event handling for the application scheduler.

The event model we propose to explore can be described in terms of three abstractions: *events*, *triggers*, and *notifiers*. An event is a tuple consisting of a timestamp, an event type, and a measurement value. A trigger is a set of constraints over events which, when satisfied, causes a set of notifiers to be invoked. The constraints are specified as a range of values that a particular resource performance profile can assume, and a boolean flag indicating whether the trigger should "trip" when an observed value falls either inside or outside the range. The VI scheduler can then specify a set of triggers for any given execution instance about which it wishes to be notified. A notifier is a function or method invoked as result of an enabled trigger. These definitions are simple and leave issues such as global time or event composition unspecified. However, we believe that they are sufficient to support VI activities. We have built a simple multi-threaded event system in C++ whose interface is tailored to the VI scheduler. We are also developing an event registry so that the scheduler can discover event sources. This builds on an ongoing effort within the Global Grid Forum [29] for defining an LDAP schema for naming and registering Grid events.

We also make use of performance forecasting when deciding to trip a trigger. On the Grid, resource performance can vary dramatically. As performance fluctuates, it is important not to burden the scheduler for many “false alarms.” For example, if an event is to be triggered when network performance drops below say, 2 megabits/sec, on a link between the University of Tennessee and University of California, San Diego, the scheduler will be notified constantly if the “normal” network performance constantly fluctuates around this value.

Forecasting, however, allows the event system to determine whether the current network reading is consistent with previously observed fluctuation. By keeping a history of previous performance measurements, and applying fast statistical analysis techniques, the system automatically identifies outlying data values. In [37] we demonstrated the effectiveness of this approach to building performance alarms for Grid resources. We are leveraging the same techniques in the VI event system.

Our initial goal in this area is to foster research results using prototypes we understand well, thus we have opted for a “home grown” event system. Distributed event systems, however, are not new. It is possible that as we go forward, we may wish to adopt an extant technology for managing distributed events. For instance, FALCON [34] provides a runtime environment specifically designed to support the steering of scientific applications. In a Grid environment, where resource performance fluctuation must be accounted for, the event system itself must be adaptive and high-performance. Our goal is to leverage the successful infrastructure we have built for monitoring and forecasting performance data in the form of the Network Weather Service (NWS) [62]. By combining extant event techniques with commonly accepted Grid technologies we will give the VI implementation of MCell the maximum flexibility we can in terms of a target execution platform.

3.4.2. Resource Access and Data

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 true because Grid computing is still an emerging technology that is not yet ubiquitous. This may change in years to come, for instance when Grid computing evolves toward a Web service architecture as proposed in [22].

The VI targets a number of middleware services, such as Globus [28], NetSolve [13], NWS [61], IBP [46].

These services can be uses simultaneously in order to expand the range of resources that can be used for 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. 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 approach to be replaced with standard Grid technology when it becomes available [29].

One of the limitations of APST is that it does not maintain persistent state about applications and resources. 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 can exploit specifics of MCell, including data management requirements, which is not possible with APST as it is general purpose. 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 presented 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.

3.5. Virtual Instrument Software

The Virtual Instrument software follows a strict object-oriented design and is constructed of three principle 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 and the daemon makes use of distributed/Grid resources.

Figure 3 depicts the interactions of the three main components of the VI architecture: the VI Daemon,

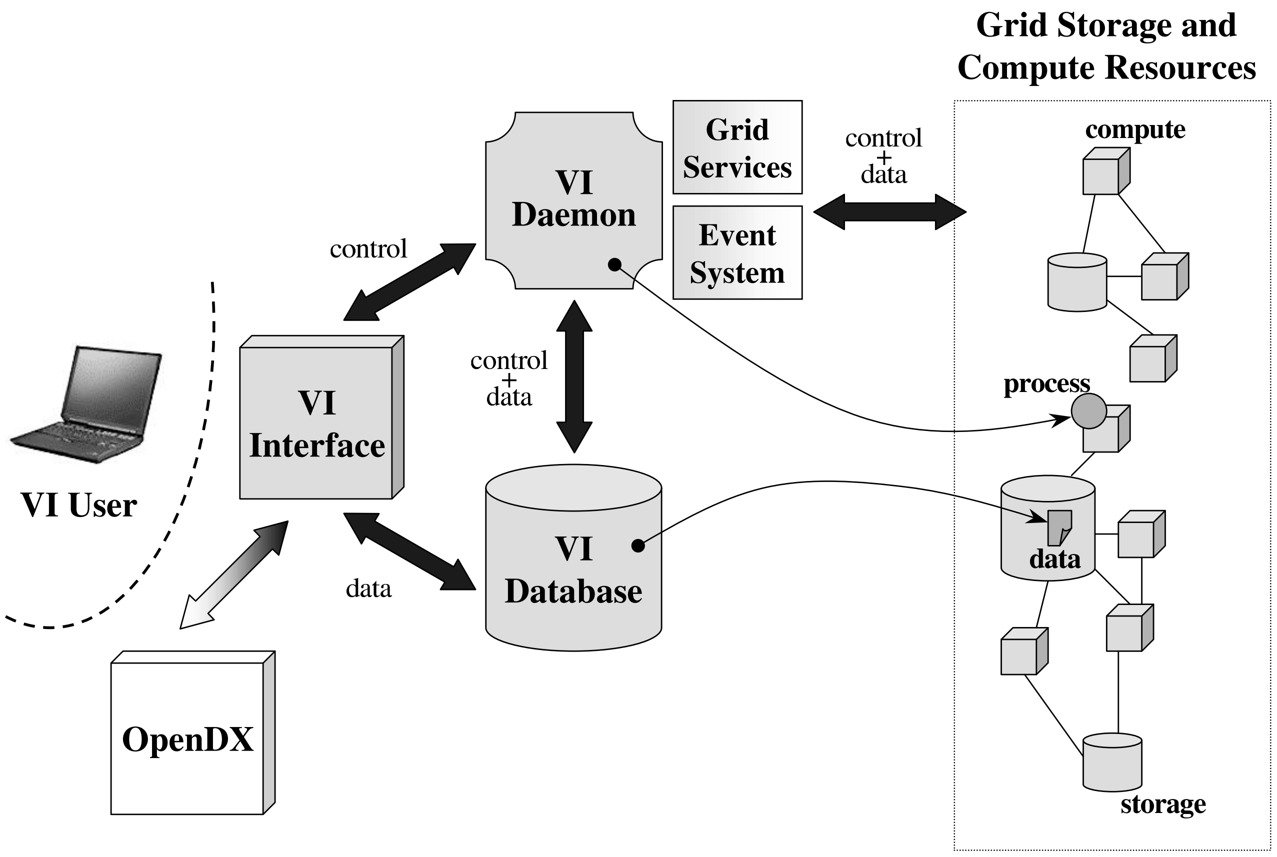

FIG. 3 The Virtual Instrument: the three main components are the VI Daemon, the VI Database, and the VI Interface.

the VI Interface, and the VI Database. The VI Daemon interacts with Grid resources and services, as well as utilizing the Event System described in Section 3.4.1. 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 VI 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. This final output data is stored in the Database by the Daemon. All raw and intermediate output is left in place in Grid storage. As depicted in Figure 3, the Database keeps track of the raw and intermediate output data for possible retrieval by the user (see Section 3.4.2).

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

The main responsibility of the VI 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 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 Grid middleware services.

The VI architecture has several key advantages over the APST design. The use of a relational database 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 this way, a user can start an MCell project, disconnect, and check the status of the simulation from any location.

Based on previous experience with APST the use of multi-threading dramatically improves the efficiency of launching tasks, since it effectively hides network and software latencies (see quantitative assessments in [15]). For that reason, each Scheduler and each Actuator runs in its own thread, and there is an independent Actuator for each resource. Throughout the VI Daemon, the Database is accessed through a common set of interfaces which streamline and automate the process of generating and executing SQL queries.

3.6. Status of the Implementation

At the moment, the VI software consists of approximatively 20,000 lines of C++. It uses pthreads and tools from the AppleSeeds libraries [3]. We opted for MySQL [40] to implement the VI Database as it is well accepted by the Linux community and provides a standard API. A later version of the VI software could use the more generic ODBC [43]. In the current release, the actuators within the VI Daemon target SSH and Globus's GRAM for starting/monitoring remote jobs, scp and GridFTP for moving application data on the Grid. Other actuators are underway (e.g. NetSolve [13] and IBP [46]). Our implementation of the VI event system targets NWS [61] for resource monitoring. The VI Interface is still underway and at the moment we provide 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. That interface is also written in C++ on top of VI components. Finally, we have implemented a simulator in order to evaluate our scheduling/steering strategies (as shown in Section 3.3). The simulator is written with the Simgrid [11] toolkit, 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.

A beta version of the VI software was released to a limited number of MCell users/developers in February 2002 for evaluation and comments. The software is making rapid progress and will be demonstrated at the SC'02 conference. More information can be found on the project's Webpage at [60].

4. RELATED WORK

Our work is related to a number of large efforts that seek to provide Grid application execution environments for scientific simulations. Like our work, those projects are targeted to specific applications or domains [33, 42, 47]. Combining the experience gathered in all those projects, given that project teams consist of computer scientists and disciplinary scientists, is critical to moving Grid technology forward. Related works also include portal activities [31, 30] and the VI software could ultimately be integrated as a user portal. Our work on an event system is related to efforts like FALCON [34], JAMM [36], as well as Grid notification activities in the Global Grid Forum [26]. Even though we opted for a custom approach for our event system, we will certainly investigate how those systems could be of benefit to the VI.

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 [35], Progress [58], Magellan [59], CUMULVS [25]). 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 of task priorities for appropriate resource sharing.

This work builds on our earlier work on the AppLeS Parameter Sweep Template (APST) [15, 4] which is related to projects such as Nimrod [1] or ILAB [63]. APST provides a generic Grid application execution environment for Parameter Sweep Applications (PSA). PSAs are applications that 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, 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. Moreover, neither APST, Nimrod, nor ILAB provide capabilities for computational steering.

5. 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 is gaining great popularity in the computational neuroscience community. Even though 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 first described our initial approach for a scheduling strategy that effectively exploits Grid resources when users can steer MCell simulations on-the-fly. We also described key contributions of our software effort, including the design and implementation of an event system, and explained how our software design targets the Grid platform and existing middleware services. Several of those contributions are relevant for Grid software development in a more general context than that of the VI project.

Many future directions are currently being explored in this project. We have highlighted only the basics of our scheduling/steering strategy and future papers will detail our approach and give many results obtained with our simulation framework. We will validate those results with actual runs of the VI software. We will also report on the implementation and effectiveness of the VI event system. In terms of software development, we will add support for additional Grid middleware systems, finalize the graphical VI Interface, and release the software to MCell users at large.

One of our goals is to deploy our software in a production environment to (i) 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.

REFERENCES

[1] 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.

[2] 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.

[3] AppleSeeds Webpage. `http://grail.sdsc.edu/projects/appleseeds`.

[4] APST Webpage. `http://grail.sdsc.edu/projects/apst`.

[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 Supercomputing'96, Pittsburgh*, 1996.

[10] 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.

[11] H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

[22] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Available at `http://www.globus.org`, 2002.

[23] 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.

[24] 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.

[25] 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.

[26] Working Group on Grid Information Services at the Global Grid Forum. `http://www.gridforum.org/1_GIS/GIS.htm`.

[27] 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.

[28] Globus Webpage. `http://www.globus.org`.

[29] Global Grid Forum Webpage. `http://www.gridforum.org`.

[30] GridPort Webpage. `http://gridport.npaci.edu`.

[31] Grid Portal Collaboration Webpage. `http://www.ipg.nasa.gov/portals/`.

[32] A. Grimshaw, F. Ferrari, A. Knabe, and M. Humphrey. Wide-Area Computing: Resource Sharing on a Large Scale. 32(5), May 1999.

[33] GriPhyN Webpage. `http://www.griphyn.org`.

[34] W. Gu, G. Eisenhauer, K. Schwan, and J. Vetter. Falcon: On-line Monitoring for Steering Parallel Programs . *Concurrency: Practice and Experience*, 10(9):673–698, 1998.

[35] 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.

[36] JAMM Webpage. `http://www-didc.lbl.gov/JAMM`.

[37] C. Krintz and R. Wolski. NwsAlarm: A Tool for Accurately Detecting Degradation in Expected Performance of Grid Resources. In *Proceedings of CCGrid 2001*, May 2001.

[38] MCell Webpage at the Pittsburgh Supercomputer Center. `http://www.mcell.psc.edu`.

[39] MCell Webpage at the Salk Institute. `http://www.mcell.cnl.salk.edu`.

[40] MySQL Webpage. `http://www.mysql.org`.

[41] H. Nakada, M. Sato, and Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 1999.

[42] National Virtual Collaboratory for Earthquake Engineering Research Webpage. `http://www.neesgrid.org`.

[43] ODBC Webpage. `http://www.odbc.org`.

[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] J. Plank, M. Beck, W. Elwasif, T. Moore, M. Swany, and R. Wolski. The Internet Backplane Protocol: Storage in the Network. In *Proceedings of NetSore'99: Network Storage Symposium, Internet2*, 199.

[47] Particle Physics Data Grid Webpage. `http://www.ppdg.net`.

[48] The Purdue University Network Computing Hubs Home Page. `http://punch.ecn.purdue.edu`.

[49] R. Rao-Mirotznik, G. Buchsbaum, and P. Sterling. Transmitter concentration at a three-dimensional synapse. *J. Neurophysiol.*, 80(6):3163–3172, 1998.

[50] M. M. Salpeter. *The Vertebrate Neuromuscular Junction*, pages 1–54. Alan R. Liss, Inc., New York, 1987. Edited by Salpeter, M. M.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] A. Torn and A. Zilinskas. *Global Optimization*, volume 350 of *Lecture notes in computer science*. Springer-Verlag, 1989.

[57] J. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:434–439, 1975.

[58] 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.

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

[60] Virtual Instrument Webpage. `http://grail.sdsc.edu/projects/vi_itr`.

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

[62] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, October 1999.

[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.