Distributing MCell Simulations on the Grid

Henri Casanova [*] casanova@cs.ucsd.edu
Tom Bartol [†] bartol@salk.edu
Joel Stiles [‡] stiles@psc.edu
Francine Berman [*] berman@cs.ucsd.edu

Abstract

The Computational Grid [21] is a promising platform for the deployment of large-scale scientific and engineering applications. Parameter Sweep Applications *(PSAs) arise in many fields of science and engineering and are structured as sets of "experiments", each of which is executed with a distinct set of parameters. Given that structure, PSAs are particularly well suited to the Grid infrastructure and can be deployed on very large scales.*

However, deployment is not easy to achieve for the domain scientist given the complexity and multiplicity of the Grid software infrastructure, the heterogeneity of the resources, and the dynamic resource availabilities. It is therefore necessary to provide user-level middleware that acts as an intermediate layer between the application and the Grid. That middleware must address all deployment, data movements, and scheduling issues to provide the user with a transparent way of running his/her simulation on the Grid.

In this paper we focus on such middleware specifically targeted to a biology application: MCell. After describing the application and its structure, we describe desired usage scenarios on the Grid. We identify user requirements, discuss relevant computer science issues and propose suitable solutions given currently available Grid technologies. We then describe a general user-level middleware project for PSAs, APST, explain how it can be extended in order to accommodate MCell's specific requirements, and introduce current work in that direction.

1. Introduction

Advances in networking technologies have made it possible to aggregate CPU, network and storage resources into *Computational Grids* [21, 28]. Such environments can be used effectively to support large-scale runs of distributed applications. A particularly important class of grid applications are science and engineering simulations, many of which can be structured as loosely coupled *Parameter Sweep Applications* (PSAs) [1, 7, 27, 41, 51]. PSAs are typically structured as sets of "experiments", each of which is executed with a distinct set of parameters. Popular examples are parameter-space searches and Monte-Carlo simulations. More specifically, we define a *PSA* as a (large) set of "independent " tasks, meaning that there is no, or little, task precedence rules. This paper focuses on MCell, a biology application which fits the PSA model and is described in detail in Section 2.

There are many technical challenges involved when deploying large-scale applications over a distributed computing environment. Although PSAs exhibit a simple structure, we will see that efficient scheduling must be employed in order to co-locate computation and data (which must be staged appropriately for good performance) in dynamic environments. Previous work [10, 44, 9] has demonstrated that run-time, adaptive scheduling is a fundamental approach for achieving performance in the context of the Grid. The development of *user-level middleware* [15] that addresses the computer science issues and targets the efficient deployment and scheduling of large-scale PSAs would be of great value for scientific and engineering community. In this paper we focus on such middleware in the specific context of the the MCell application.

This paper is organized as follows. Section 2 describes MCell, its current implementation, and details requirements for running large-scale MCell simulations. Section 3 identifies the computer science issues in meeting the aforementioned requirements and proposes a number of solutions. Section 4 describes current work being done as part of APST, a user-level middleware software project targeted to PSAs. Finally, Section 5 highlights related work and Section 6 concludes the paper.

[*] CSE Dept., University of California, San Diego
[†] Computational Neurobiology Laboratory, Salk Institute
[‡] Biomedical Applications Group, Pittsburgh Supercomputing Center

2. MCell: Application and Requirements

2.1. MCell Overview

MCell [29, 30, 50, 47] stands for "Monte Carlo cell" or "MicroCellular physiology". In general MCell 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 the 3D diffusion space(s), which then can be populated with molecules of different kinds [48]. 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 that recapitulates net displacements arising from Brownian motion during time-step intervals. During each "walk", possible reaction events such as binding and unbinding are tested on a molecule-by-molecule basis using random numbers and Monte Carlo probability values [46]. Section 2.2 provides an example of MCell's use for quantitative simulations of synaptic transmission.

In an MCell simulation, molecular diffusion and reactions are modeled using a stochastic treatment of phenomenological rates, and the simulation time-step can often be on the microsecond scale for signals lasting from milliseconds to seconds. MCell methods and algorithms thus are positioned "above" those of molecular dynamics, which compute atomic forces of interaction to predict detailed structure and function of single molecules, and which generally require a time-step on the femtosecond scale. On the other hand, MCell simulations are situated "below" models that compartmentalize whole cells (or groups of cells) and simulate processes within and interactions between compartments by using coupled sets of differential equations. Important distinguishing features of MCell simulations therefore include the development and use of highly realistic 3-D biological structures, and realistic movements and interactions of individual molecules within the structures. Since decisions underlying molecular events are made using random numbers and probabilities, the simulation results accurately reflect biological "noise" that may have functional consequences and can show a complex dependence on space and time.

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

1. **Surface design or reconstruction.**
   In simple cases, a plane or small 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.**
   A set of surfaces is only the starting point for a realistic model, since 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. It may also be of interest to alter the surfaces and/or molecular distributions in some systematic or arbitrary way, e.g., to simulate the pathogenesis of a disease that perturbs one or more aspects of subcellular organization.

3. **Simulation of the signal or metabolic process.**
   Often this step involves some form of parameter sweep. In one scenario, simulations are repeated for different values of one or more input parameters, to define how one or more output values change with the changing inputs. In another scenario, simulations are repeated with input parameters varied according to some approximate fitting scheme, so that output values ultimately match (within some confidence range) a corresponding set of experimental target measurements. In either case, a particular set of input parameters may need to be used in multiple simulations run with different streams of random numbers, in order to quantify the predicted noise and/or reduce it by averaging the output results. As discussed in more detail in Section 2.3, the total number of simulations required for present modeling projects typically ranges anywhere from 10^2 to 10^5 and beyond.

4. **Visualization and analysis of results.**
   In the simplest case this might be a 2-D plot 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 simulated signal's evolution in time and space, as well as multi-dimensional relationships between input and output parameters.

Typical events that may occur during each time-step of an MCell simulation include:

- the release of diffusing molecules from some structure defined by a surface,
- random walk movements for diffusing molecules,
- encounters between diffusing molecules and reflective, transparent, or absorptive surfaces,
- creation of new diffusing molecules at some point or points in space,

- chemical reaction transitions undergone by single molecules or two molecules that encounter each other during diffusion,
- counting of molecules and reaction transitions in various time- and space-dependent ways, and
- output of surface and molecule information for visualization.

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.

Each time that a diffusing molecule makes a random walk movement, its trajectory must be traced to determine whether it intersects with another object before reaching the anticipated endpoint of motion. If an intersection does occur, a random number is used to decide between all possible events, and then the search for collisions along the original or modified trajectory must continue as necessary. This process of marching along a trajectory and making decisions accounts for most of the computer time required for the simulation.

To optimize the search for intersections, the simulation space can be divided into subvolumes, each of which contains only a small number of objects (under optimal conditions, subject to memory limitations). By searching and traversing subvolumes rather than the total space during each time-step, the computer time required for a simulation can be nearly independent of the model's spatial complexity. For large- scale models the effective speed-up amounts to many orders of magnitude, e.g., the difference between minutes and weeks. In the absence of such run-time optimizations, MCell simulations of realistic structures would routinely require tightly coupled processing on massively parallel architectures. With such optimizations, however, simulation conditions are often loosely coupled or embarrassingly parallel, so that Grid-based computation becomes eminently feasible.

2.2. MCell Simulations of Synaptic Transmission

Nerve cells communicate with themselves and other cells at structures called *synapses*. The synapse itself is defined by a presynaptic cell in which the signal originates, a postsynaptic cell to which the signal must be transmitted, and an intervening gap, or synaptic cleft. The signal is carried by thousands of small neurotransmitter molecules (e.g., amino acid molecules) that are released in discrete packets from the presynaptic cell, diffuse across and within the cleft, and activate postsynaptic receptor molecules (typically one or more varieties of protein molecules). Activation of the receptors initiates a tiny electrical and/or chemical signal that can be measured experimentally.

The synaptic cleft is simply a specific region of the extracellular space that surrounds and separates all cells from each other. In some cases the cleft architecture is relatively simple, e.g., like a coin that is about 15 nanometers high and several hundred nanometers across, but its edges may communicate with an extensive and tortuous diffusion space defined by the surrounding cells. In other cases, the cleft architecture itself is extremely complex, with many folds, twists, and tunnels from one region to another. With present molecular biological techniques, it is increasingly possible to determine the density and distribution of receptors and other important molecules within the cleft and surrounding diffusion space. It is also increasingly possible to determine the chemical reaction pathways and rates that determine how the neurotransmitter molecules interact with the receptors, and how the resulting signal may initiate a cascade of postsynaptic events. When this information is paired with a 3-D reconstruction of the synaptic surfaces, it becomes possible to build a highly realistic model of the synapse, which then can be used in MCell simulations of synaptic transmission. Quantitative predictions for the synaptic signal and signal variability then can be compared directly with experimental data, to address such questions as:

1. How does the architecture of the synaptic cleft affect the size and time course of single and multiple synaptic signals in different regions of the brain, or at synapses elsewhere in the body?
2. How do the kinetics and precise localization of transmitter release, receptors, and transmitter removal from the synaptic cleft influence signal variability and the spread of transmitter from one synapse to another (synaptic cross-talk)?
3. How do activity-dependent changes in synaptic structure and function contribute to cognitive processes such as learning and memory?

MCell is an outgrowth of combined experimental and computational studies originally focused on the vertebrate neuromuscular junction, the synapse between a nerve cell and a muscle cell (reviewed in [42] and [48]). The neuromuscular junction is the first synapse to have been studied in molecular detail, and over many years a wealth of structural and physiological data has been amassed. Thus, this synapse has been particularly well suited to increasingly realistic and ongoing Monte Carlo simulations aimed at:

1. Quantitative reproduction of synaptic signals produced by individual packets of the neurotransmitter acetylcholine [4].

2. The sensitivity of synaptic signals to the density and activity of a synaptic enzyme that binds and destroys acetylcholine [3].

3. The mechanism by which acetylcholine is released and its impact on the rising phase of the synaptic signal [50, 47, 48].

4. The biophysics of acetylcholine receptor activation and the sensitivity of synaptic signals to temperature and variation of numerous kinetic parameters [49].

5. The influence of highly realistic synaptic architecture on signal variability [48].

6. Large-scale parameter sweeps and sensitivity analyses (Section 2.3).

The earliest projects listed above were conducted with predecessors of MCell, programs that were tailored to the problem at hand and could not be applied to larger, more realistic models of the neuromuscular junction or other synaptic systems. Such limitations were eliminated for MCell by generalizing the underlying Monte Carlo algorithms, developing numerous run-time optimizations (including the use of spatial subvolumes as outlined in Section 2.1), and creating the MDL. Present scientific projects in the development groups include continuations of those listed above, as well as new large-scale reconstructions and simulations of synapses in the mammalian brain, and normal, diseased, and mutant neuromuscular junctions in mammals, vertebrates, and invertebrates. In addition, MCell has been in limited release [29, 30] to a worldwide group (~25) of Neuroscience and other research laboratories since 1997 [24, 40, 19, 18], and new algorithms designed to extend its simulation capabilities remain under active development. A general release planned within several years will likely coincide with release of the Grid computation middleware outlined here.

2.3. 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 resource requirements – from very simple "look & see" cases run on a single workstation, to to very large-scale models with many diffusing molecules and large memory requirements distributed across nodes of a parallel machine, and/or to sweeps of high-dimensional parameter spaces run on the Grid or a massively parallel supercomputer.

Computational experiments conducted using MCell thus can be categorized according to the scale of each individual simulation (number of iterations, memory requirement, number of diffusing molecules, size of input files, size of output files, etc.) and the number of individual simulations to be performed. A number of increasingly common usage scenarios include:

A. **"Look & See" Scenario.**
(To paraphrase Yogi Berra – "You can see a lot by looking"). A single simulation fits in available RAM and runs in less than one day on a typical workstation. Only a small number of runs are used to determine the predicted behavior of the modeled system under some set of input conditions. Note that the input parameters may include variables related to spatial structure as well as others related to the chemical reaction network.

B. **Parameter Fitting Scenario**
A single simulation fits in available RAM and runs in less than one day on a typical workstation or single processor of a parallel machine. Depending on the dimensionality of the parameter space, tens to thousands of runs may be required to identify a set of input parameter values which produce model output that matches some set of target criteria. As in the "look & see" scenario, input parameters may include structural as well as reaction network variables.

C. **Parallel MCell Scenarios**
A single simulation includes too many molecules or requires too many iterations to be run in a reasonable amount of time on a single processor -*or*- requires too much RAM to fit on a single workstation. A parallel version of MCell is used (see below) on a parallel platform in a "look & see", parameter fitting, or parameter sweep scenario mode.

D. **Parameter Sweep Scenario**
The scale of a single simulation is similar to that for the parameter fitting scenario, but many thousands of runs are required to map a region of the multidimensional parameter space encompassed by the model.

As a particular MCell model evolves and matures (due to feedback between earlier model results and comparisons made to expectations and results gleaned from the literature), it is likely that usage will migrate from a "look & see" scenario (A) to some scale of parameter fitting and/or a full-blown parameter sweep scenario. If individual runs fit in available RAM and require less than one day, the parameter fitting and sweep scenarios (comprising thousands of independent runs) become well-suited to implementation on Grid resources. However, consideration should be given to the size of the input data required and output data generated by each run. A Grid implementation can gain much

efficiency by using resource discovery techniques and subsequently staging large input files on storage resources close to the scheduled computational resources.

In the case of the parameter fitting scenario (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. Under this scenario MCell use becomes a combination of "high throughput computing" and "on demand computing"[21], and an MPP/reservation/batch model is not desirable. Instead, one would prefer even a dynamic (shrinking/expanding) set of resources where one can at least be sure that "some" computation will be performed, rather than waiting in a queue for hours only to obtain intermediate results.

For very large individual simulations a parallel version of MCell (C) (implemented using PVM or MPI, for example) must be used to obtain adequate throughput. An initial parallel version of MCell currently subdivides the computation spatially, i.e. each processor is given all of the structural and molecular components contained within a given subvolume of the space. Thus the required memory is distributed as is the computation associated with the diffusing molecules. Diffusing molecules that cross the spatial partition between two processors are handled using message passing. The parallel MCell usage scenarios (item C above), especially a parallel MCell parameter sweep scenario, will be a future part of Grid computation projects as the large-scale parallel resources on the Grid continue to grow.

A concrete example of a parameter sweep scenario (D) is provided by a recent computational experiment conducted using BlueHorizon (an 1152 processor IBM SP supercomputer at SDSC) and a Linux cluster in Japan (256 933MHz processors). The experiment involved mapping a portion of a 4-D kinetic parameter space included within a model of synaptic transmission at a simplified neuromuscular junction [5]. The parameter space was sampled at 4704 locations. To reduce stochastic noise each set of input conditions was repeated 20 times using different streams of pseudorandom numbers, and the results were averaged.

The MDL input files for this simplified neuromuscular junction model consisted of about 5KB of data. The raw output of each simulation was a series of numerical values (a time series) stored in a file of approximately 1MB. Thus, the experiment produced a total of 94080 raw output files, or about 94GB, which after averaging condensed into 4704 time series (about 4.7 GB). Subsequent analysis of the averaged time series files then ultimately yielded about 600KB of additional output data. The averaging and analysis steps took only a few seconds per task. However, the compute time for each simulation averaged about 30 minutes, and varied from 10 seconds to 6 hours as input parameter values changed dramatically. It is important to note that the input files to this experiment were very small. In general, input files to MCell can range from a few KB for simple structures to tens of MB or more for realistic reconstructions. This fact combined with the large amount of simulation output makes databases and effective data management strategies an important future focus.

3. Challenges to Running MCell Simulations on the Grid

In this section we identify specific issues and challenges that must be considered when designing and building user-level middleware for deploying MCell on the Computational Grid (scenario D). We describe possible solutions and highlight shortcomings of currently available Grid software infrastructures.

3.1. Deployment

Many services and mechanisms are needed for deploying large-scale applications over widely distributed resources that are shared and often span multiple administrative domains. We distinguish here 5 types of services that are necessary to deploy MCell over the Grid:

(i) resource discovery,
(ii) software dissemination,
(iii) resource access,
(iv) job control and monitoring,
(v) distributed storage.

Resource discovery allows an application/user to acquire information about potential resources and is usually implemented by providing well-defined information services. Software dissemination is enabled by mechanisms for the user to use for easily installing and running his/her application software on remote Grid resources. Resource access involves security issues (authentication/encryption) as well as mechanisms to place calls to the application software (APIs, IDLs). Job control and monitoring is used to detect failure and possibly measure the progress of application tasks.

Finally, distributed storage infrastructures provide means of managing application data. The efforts underway in the Grid Forum (GF) activity [28] aim at generating a specification for most Computational Grid services, and at making recommendations about implementation choices. The Grid Forum standardization effort is a long term process, and our goal is to deploy MCell simulations now. Consequently, our approach is to use existing technologies that already provide prototype implementations of one or more of the aforementioned services. We expect an easy transition to the standard Grid infrastructure as it becomes increasingly available.

Resource discovery is usually based on deployed and ubiquitous information services. Globus provides such a service in the form of the LDAP-based MDS [20]. Legion [25] provides its own information service. Using the LDAP protocol is the current trend in *Network-enabled Servers* projects [14], is advocated by GF, and should be used by user-level middleware such as the one we envision for MCell. **Software dissemination** is not supported by many Grid research softwares. The Globus Executable Management (GEM) toolkit component is not available, and other projects [12, 37] require the user to pre-install software "manually" on remote systems. Legion on the other hand provides a transparent way of matching and uploading binaries to appropriate architectures. This is the sort of mechanisms that should be used by the middleware when they become part of a Grid standard.

The security aspect of **resource access** is PKI-based in Globus, Legion, and other projects [37]. The Globus Security Infrastructure (GSI) [23] is becoming the de-facto standard implementation and current work focuses on allowing Legion to run on top of GSI. User-level middleware for MCell will use the GSI. As far as access mechanisms, the Network-enabled Server paradigm provides the programming model that is the most adapted to MCell's needs: enhanced RPC-style calls, which is higher-level than but similar to the Globus GRAM [17] mechanisms. RPC-style programming will be the underlying programming model for MCell on the Grid.

Given the structure of MCell simulations, **job control and job monitoring** mechanisms need not be sophisticated. Simple failure detection tools like the Globus HBM [45] or the internal mechanisms of projects like NetSolve [12] as they are available today are quite sufficient. Job monitoring in terms of performance (or measure of progress) would also be valuable but should probably be implemented as part of the MCell software itself. Finally, there is a pressing need for a Grid **distributed storage** infrastructure. MCell manipulates tremendous amounts of data that are scattered over a large number of files. It can therefore benefit from mechanisms to name, locate, duplicate, and efficiently disseminate that data. The infrastructure proposed in [16] promises to address all these issues but none of the currently available projects/protocols provides such solutions. At the moment, an implementation of user-level middleware for MCell needs to leverage multiple technologies [22, 39, 32] and must implement its own data abstraction layer.

Section 4 presents a prototype user-level middleware for PSAs, APST, and details the design and implementation choices that prevailed for deploying applications with current Grid technology. The experience gained while implementing APST should be profitable to the design of a next-generation middleware specifically targeted to MCell.

3.2. Scheduling

MCell simulations are structured as sets of Monte-Carlo simulations, implying that tasks are independent, and therefore indicating that scheduling should be straightforward. However, the problem of scheduling sets of independent tasks onto heterogeneous sets of processors has been identified as NP-complete [34] and substantial research work has been devoted to design suitable heuristics [26, 33, 34, 36, 11]. These heuristics generally fall into two categories: those which use *self-scheduling*, and those which use *performance prediction*. While the former are easy to implement and extremely adaptive, the latter aim at making a *plan* for the application execution and should be able to take advantage of idiosyncrasies of the application and the computing environment.

In the case of MCell, two additional aspects of the structure of simulations complicate the scheduling problem. First, independent tasks may share large input files and, depending on the available compute and storage resource, it may become critical to place such files in strategic locations and then to co-locate computation and data. Second, multiple post-processing stages of the raw output data are needed to present the MCell user with synthesized results that are needed for scientific interpretation. As seen in Sections 2.2 and 2.3, the post-processing often includes averaging multiple time series that can reside in large output files. It might then be desirable that these files be located in proximity to each other and an effective scheduling algorithm should ensure that post-processing is done efficiently.

Work in [13] addressed the question of scheduling PSAs over a heterogeneous Grid that consists of multiple *sites* that contain compute resources (hosts) and a data repository that can be shared among those resources (e.g. a disk over NFS). More specifically, the focus in that paper was to determine which heuristics are appropriate to ensure that input files are shared efficiently among application tasks. Simulation results showed that adaptive versions of heuristics using performance prediction are the best solution even in the presence of prediction inaccuracies. In particular, the *XSufferage* heuristic (which schedules the task first that would "suffer" most if scheduled badly) is very promising. An implementation of these heuristics is provided in the APST project (see Section 4) and requires support for resource performance monitoring and forecast. Our experiments with MCell on real testbeds used the Network Weather Service (NWS) [52] as it is currently deployed and used in multiple institutions and provides performance-efficient predictions.

The work presented in [13] is limited in its application to MCell in two ways. First, the scheduling heuristics are not designed to take into account the output post-processing stage and to date, there has been no evaluation of their be-

haviors in such a scenario. Second, the model for the computational environment is restricted to one data repository per site, with no file exchanges among sites (the authoritative source of application data files is the user's host). To accommodate MCell simulations on real Grid testbeds we must design new heuristics that take into account output post-processing and evaluate them in simulation with a more general setting where data repositories can be located anywhere, and where file transfers among any repositories are allowed.

To address this, we are currently considering two approaches. A straightforward approach would be to modify the existing heuristics so that application tasks generating output files that need to be combined are scheduled on compute resources "close" to a single data-repository. This however does not provide a general solution that will be effective in multiple Grid settings or for all application scenarios (e.g. "on demand" computing). It is the case that the averaging of MCell time series is an associative operation and can be therefore performed in a "divide-and-conquer" fashion. An alternate approach is then to perform iterative post-processing of the output according to the respective location of output files. These two approaches are fundamentally different in that the former aims at imposing the location of output files in advance, whereas the latter tries to optimize output post-processing given the location of output files. Our future work will be focused on evaluating these two approaches as well as designing and evaluating heuristics that provide a middle-ground.

A current shortcoming of current Grid monitoring systems is that network topologies are usually not accurately taken into account when providing performance metrics concerning point-to-point data transfer rates. Typically, the monitoring systems perform series of point-to-point benchmarks *independently* which does not reflect possible link contention. There is no indication of what data transfer rates would be observed if *simultaneous* data transfers are initiated. Some projects [35, 43] address the modeling/monitoring link contention but are not widely deployed. Unfortunately, the more general Grid model and the modeling of associative collective operation that we discussed earlier must rely on network performance evaluations that take link-contention into account. Even though the work in [13] presents an adaptive scheduling algorithm which is tolerant to performance prediction inaccuracies, we expect that a reasonable estimation of the impact of link contention on data transfer rates will be necessary for effectively scheduling the output post-processing stages in MCell. Our first step will be to use simulation to evaluate new heuristics for scheduling MCell with output post-processing and over a general Grid topology. We then plan to implement the best heuristics as part of a user-level middleware such as the one presented in Section 4. That implementation will initially use ad-hoc techniques to assess link contention, until more sophisticated Grid monitoring services are deployed.

3.3. User Interface

The ideal user interface for MCell simulations must encompass each of the four steps outlined in Section 2.1, which means it must provide an interactive visual environment for model creation and editing, simulation control, and post-processing of multi-dimensional output data. For Grid-based computation, it must also provide interactive access to the middleware tools that will deploy and monitor large sets of MCell tasks running under the Parameter Fitting and Parameter Sweep usage scenarios outlined in Section 2.3.

Not surprisingly, the actual existing user interface is fairly far removed from the ambitious ideal outlined above, and this reality arises largely from three factors [48]: first, the relatively recent expansion of MCell's algorithms and optimizations to handle large-scale polygon mesh objects, and therefore highly realistic cellular models; second, a general lack of surface reconstruction software well-suited to the highly accurate and unflawed large-scale tessellations required by MCell models; and third, the general complexities and computational load imposed by large-scale interactive 3-D rendering. Aside from MDL text files, the existing interface thus comprises a loosely knit collection of open source, custom written, and commercial tools and scripts for 2-D pixmap and vector graphics, surface generation and optimization, interactive 3-D rendering of down-sampled models, batch-mode rendering of full-scale models and time series animations, and interactive multi-dimensional visualization of post-processed output results.

To assimilate the diverse aspects of an MCell user interface into one cohesive yet flexible package, we plan to use OpenDX [31], an open source outgrowth of IBM Data-Explorer software. OpenDX is a powerful interactive analysis and rendering tool for multidimensional datasets, and is based on a very general data model for volumes, surfaces, and arbitrary data properties and interdependencies, i.e., is highly compatible with an emerging and expanding MCell data model for cell surfaces and surface properties. OpenDX also includes an extensive visual programming environment that can be used as a graphical interface builder and to link operations to external applications (e.g., the APST daemon discussed in Section 4), and already has been used for many of the most important stages of MCell model design. To whatever extent the native visual programming toolset proves inadequate for a general MCell interface, custom modules can be written and added, and if necessary, a Java-based front-end can be employed for further customization.

As explained in Section 2.3, MCell-specific data management strategies are needed for maintaining databases of

both MCell input and output data. Ideally, the MCell user interface would allow the user to easily archive, access, and transform that data. We are currently refining the data management model for generic MCell simulations and implementing that model in structured databases. We will then write an OpenDX module that interacts with the databases. Ultimately, we envision the use of OpenDX as a unified tool for the entire life-cycle of an MCell project: input data preparation, simulation runs for the four scenarios described in Section 2.3, output data processing, analysis and visualization.

4. A Prototype User-Level Middleware: APST

In this section we introduce the *AppLeS Parameter Sweep Template* (APST), a prototype user-level middleware project that targets PSAs in Computational Grid environments. An experimental evaluation of APST is presented in [15]. The goal of APST is to provide the end-user with easy and efficient access to many Grid resources that are currently available through various software infrastructures. Performance is the responsibility of an embedded scheduler that works "behind the scene" on the user's behalf. Given these goals, we highlight relevant aspects of the APST software design and describe the current scheduling algorithm implementations.

The APST software is composed of a *client* and a *daemon*. The client is an executable that takes various command-line arguments and can be used by the user to interact with the daemon: submitting new computational tasks, cancelling tasks previously submitted, and inquiring about the status of an ongoing simulation. To submit new tasks, the user must provide a *task description file* which contains one task description per line. Each task description specifies which program to run as well as the required command-line arguments, the location of input files, and where output files should be created (i.e. written to local disks or left in place in remote storage).

The APST daemon consists of four distinct sub-systems: a Controller, a Scheduler, an Actuator, and a Meta-data Bookkeeper. Each sub-system defines its own API. Those APIs are used for communication/notification among subsystems. Providing multiple implementations of these APIs makes it possible to *plug in* different functionalities and algorithms into each APST sub-system. For instance, writing multiple implementations of the Scheduler's API is the way to provide multiple scheduling algorithms.

The **Scheduler** is the central component of the APST daemon. Its API is used for notification of events concerning the application's structure (new tasks, task cancellations), the status of computational resources (new disk, new host, host/disk/network failures), and the status of running tasks (task completions or failures). The behavior of the scheduler is entirely defined by the implementation of its API. The **Controller** relays information between the client and the daemon and notifies the Scheduler of new tasks to perform or of task cancellations. It uses the Scheduler's API and communicates with the client using a simple wire protocol. The **Actuator** implements all interaction with Grid infrastructure software for accessing storage, network, and computation resources. It also interacts with the Grid security infrastructure on behalf of the user when needed. There are two parts to the Actuator's API: one for file transfers and storage, and one for task launching, monitoring, and cancellation. The Scheduler can place calls to both these APIs to make the Actuator implement a given schedule on Grid resources. The Actuator's implementation makes use of standard interfaces to Grid infrastructure softwares to interact with resources. Each sub-system API consists of less than 15 functions.

Our design ensures that it is possible to mix and match different implementations of the internal APIs. In particular, the implementation of a given scheduling algorithm is completely isolated from the actual Grid software used to deploy the application's tasks. The intent of our design is that a constant control cycle between the Scheduler and the Actuator is necessary for effective scheduling. The Scheduler periodically polls the Actuator for task completions, failures, or other events (e.g. newly available computing resources). This leads the Actuator to place calls to the Scheduler's API to notify the Scheduler of these events. The Scheduler has then the opportunity to react by making decisions and placing calls to the Actuator's API. Such a design makes it very easy to implement straightforward algorithms like a self-scheduled workqueue, as well as more complex algorithms as the one that were introduced in [13].

An prototype implementation of APST was demonstrated during the SuperComputing'99 conference (running over 200 hosts all over the world). A beta version has been released and is being tested and evaluated for different applications and in various testbeds (e.g. NASA's IPG). Interfaces to Globus' GRAM [17] and GASS [22], NetSolve [12], IBP [39], and the NWS [52] have been implemented. A Condor [6] interface is underway. The scheduler can use six different scheduling algorithms. Among those are two self-scheduling algorithms: *wq* which is a standard workqueue, and *wq+* which is a workqueue with task duplication and task priorities. The four other scheduling algorithms use performance predictions. Three of them use the heuristics presented in [36] (*MaxMin*, *MinMin*, and *Sufferage*) and one uses a new heuristic, *XSufferage*, that was introduced in [13]. The overall design provides great flexibility as it is possible to mix and match implementations. The APST daemon can simultaneously use Globus and NetSolve servers for tasks, as well as IBP, GASS, and NFS servers for storage. Note that using distributed stor-

Table 1. Available implementations of components of the APST daemon

APST Component	Available Implementations
Actuator/Data	GASS, IBP, NFS, NetSolve
Actuator/Computation	GRAM, NetSolve, (Condor)
Meta-Data Bookkeeper	NWS
Scheduler	wq, wq+, MinMin, MaxMin, Sufferage, XSufferage

age systems as GASS and IBP allows for more flexible scheduling of tasks and data and thereby makes it worthwhile to use scheduling algorithms based on performance prediction [13]. Table 1 shows a summary of the available APST module implementations (ones that are underway are shown in parentheses).

APST in its current form is suitable for running large-scale MCell computations. In fact, it was used successfully to run MCell simulations (15,000 tasks, 20GB of output data) and proved very valuable in that it hides all logistic details from the user while ensuring reasonable scheduling. The testbeds being used for those experiments were large and distributed enough to observe the respective performance of different scheduling algorithms [15]. Our goal is to build on APST to generate MCell-specific user-level middleware. That middleware will focus on those issues highlighted in Section 3 that are not currently addressed by APST: scheduling for output-post processing and for more general Grid topologies; and MCell-specific user interface with visualization capabilities. As seen in Section 6, we are currently working on such a middleware project that will also provide *computational steering* capabilities.

5. Related Work

Parameter Sweep applications occur in many fields of science and engineering. A few project provide the necessary computing infrastructure to deploy these applications on large sets of resource. For example is the work in [8] describes Condor [6] support for Monte-Carlo simulations. In terms of Grid middleware, we gave the example of APST [15], and one can also cite Nimrod/G [2] and SciRun [38]. Nimrod/G is related the most to the work described in this paper as it targets PSAs. It provides a generic user interface that is more evolved than the one provided by APST. There are however two major differences between Nimrod/G and our work. First, the scheduling approach in Nimrod/G does not take into account the co-scheduling of computation and data as it does not use distributed storage infrastructures. Consequently, the scheduling algorithms in Nimrod/G do not specifically address the post-processing of large amount of application output data such as what is generated by MCell simulations. Second, there is no built-in way to add an application-specific user interface on top of Nimrod/G and the existing interface does not meet the MCell requirements described in Section 3.3. By contrast, the project described in this paper focuses on scheduling algorithms and on an MCell user interface that will be developed in collaboration with MCell developers and users.

SciRun does not specifically target PSAs but provides support for runtime visualization of scientific simulations. In addition it addresses computational steering issues which is not currently done in either Nimrod or APST. SciRun will be a great source of inspiration to develop the proposed user interface. Furthermore, our ultimate goal is to implement a middleware that has the capability to steer MCell computations and we will additionally benefit from the SciRun experience.

6. Conclusion and Future Work

The Computational Grid [21] is an attractive platform for running large scale application as it provides tremendous compute, networking, and storage resources. However, its complexity makes it difficult for end-users (e.g. domain scientists) to deploy their applications. It is thus necessary to develop *User-Level Middleware* which performs logistical tasks on behalf of the user, provides an attractive user interface and promotes performance by using appropriate scheduling algorithms. In this paper, we focused on middleware that is specifically targeted to support performance-efficient, large-scale runs of MCell, a biology parameter sweep application. After introducing MCell and its computational structure, we identified requirements for the desired usage scenario of MCell on the Grid. We gave implementation solutions given the current state-of-the-art of Grid technology. We then described preliminary work on a middleware project that targets generic parameter sweep applications: APST. We pointed out what additional features and capabilities are needed for MCell-specific middleware and explained how the work in APST can be re-used to that end. The two main areas to expand on are *scheduling* and *user interface* in order to meet MCell's specific requirements.

In addition, ongoing research involving the authors as well as a number of other researchers aim at providing the additional capability of computational steering for MCell. That research will be the object of a future paper when a first prototype is available. Middleware which includes the functionality described herein as well as computational steering will make MCell truly usable on the Grid and provide its users with tremendous computational power, as well as a new level of convenience and feedback. We hope that MCell with this functionality will become one of the first

Grid applications to be routinely deployed and widely used by a broad community of users and domain scientists, and will make possible the discovery of a new generation of disciplinary results.

References

[1] D. Abramson, M. Cope, and R. McKenzie. Modeling Photochemical Pollution using Parallel and Distributed Computing Platforms. In *Proceedings of PARLE-94*, pages 478–489, 1994.

[2] D. Abramson, J. Giddy, I. Foster, 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*, May 2000. to appear.

[3] 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.

[4] 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.

[5] T. M. Bartol, T. J. Sejnowski, B. R. Land, E. E. Salpeter, and M. M. Salpeter. A sensitivity analysis of chemical kinetics parameters for the neuromuscular junction. In *Soc. Neurosci. Abstr.*, 2000.

[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] J. Basney, M. Livny, and P. Mazzanti. Harnessing the Capacity of Computational Grids for High Energy Physics. In *Conference on Computing in High Energy and Nuclear Physics*, 2000.

[8] J. Basney, R. Raman, and M. Livny. High-throughput Monte Carlo. In *Proceedings ot the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, March 1999.

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

[10] 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.

[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 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.

[13] 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.

[14] H. Casanova, S. Matsuoka, and J. Dongarra. Network-Enabled Server Systems: Deploying Scientific Simulations on the Grid . 2001. submitted to HPC'01.

[15] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of SuperComputing 2000 (SC'00)*, Nov. 2000. to appear.

[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. *the Journal of Network and Computer Applications*, 2000. to appear.

[17] 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.

[18] 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.

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

[20] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, 1997.

[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. Tedesco, and S. Tuecke. GASS: A Data Movement and Access Service for Wide Area Computing Systems. In *Proceedings of the Sicth workshop on I/O in Parallel and Distributed Systems*, May 1999.

[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] 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.

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

[26] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[27] S. Hill. *Spatial and temporal processing in thalamocortical neural networks*. PhD thesis, University of Lausanne, Switzerland, 1999.

[28] http://www.gridforum.org.

[29] http://www.mcell.cnl.salk.edu.

[30] http://www.mcell.psc.edu.

[31] http://www.opendx.org.

[32] http://www.webdav.org.

[33] 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.

[34] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonindentical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[35] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A Resource Query Interface for Network-Aware Applications. In *Proceedings of the 7th IEEE Smposium on High-Performance Distributed Computing*, July 1998.

[36] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. Freund. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In *8th Heterogeneous Computing Workshop (HCW'99)*, pages 30–44, Apr. 1999.

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

[38] 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.

[39] 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.

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

[41] S. Rogers. A Comparison of Implicit Schemes for the Incompressible Navier-Stokes Equations with Artificial Compressibility. *AIAA Journal*, 33(10), Oct. 1995.

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

[43] G. Shao, F. Breman, and R. Wolski. Using Effective Network Views to Promote Distributed Application Performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[44] N. Spring and R. Wolski. Application level scheduling: Gene sequence library comparison. In *Proceesings of ICS'98*, July 1998.

[45] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. Fault Detection Service for Wide Area Distributed Computations. In *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*, pages 268–278, 1998.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, pages 97–104, Aug 1999.

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