Network-Enabled Server Systems: Deploying Scientific Simulations on the Grid

Henri Casanova * Satoshi Matsuoka † Jack Dongarra ‡

October 2, 2000

Abstract

The Computational Grid [1] is a promising platform for running large scale scientific applications. It provides a base software infrastructure that allows for the development of "middleware" aimed at deploying applications on Grid resources. The Network-Enabled Server (NES) paradigm is a good candidate as a viable Grid middleware that offers a simple yet powerful programming model (RPC-style programming for the Grid). This paradigm is amenable to many large-scale applications and especially to scientific simulations. This paper builds on the experience acquired while building two well-known NES systems (Ninf [2] and NetSolve [3]). Our goal is to clarify major NES design issues as well as to define a common set of services and concepts that are necessary for implementing and deploying NES systems on the Computational Grid. This paper also describes current work with scientific and engineering simulations that are enabled by NES systems in the Grid context.

1 Introduction

Several projects [2, 3, 4, 5, 6, 7] aim at providing simple ways (APIs, GUIs) to execute software remotely on the Computational Grid. Typically the software executed by servers consists of scientific libraries or programs. In what follows, we use the term *module* to denote that software, whether it is part of a library, a stand-alone executable, a set of executables, or any combination of the above.

This paradigm has commonly been labelled *Network-Enabled Server* (or NES) and each aforementioned project offers variations on the popular Remote Procedure Call (RPC) theme. For instance, an intelligent scheduling agent that makes decisions about task/resource assignment can intercept the RPC call, or the output data can be left in place so that it can be retrieved at a later time, etc. The common concern of all currently available NES systems is *easy-of-use*. Indeed, target users are not computer scientists but domain scientists and the NES system is supposed to hide most of the logistical details and present the set of available remote resources as a cohesive multi-purpose machine where a wealth of scientific software is available. Furthermore, different users have different needs and NES systems often strive to provide multiple interfaces and APIs (e.g. C, Java, Matlab, Fortran, Web, e-mail, MS EXcel, etc.).

Even though recent developments in Ninf [2] and NetSolve [3] provide promising extensions to the programming model (e.g. run-time RPC calls data-dependencies analysis), this is kept somewhat restricted as it is key to providing straightforward user interfaces and to making the systems ready to use out-of-the-box. Fortunately, many scientific applications are amenable to the RPC-style programming model, particularly scientific simulations (e.g. Monte-Carlo simulations), engineering design problems, and in general parameter search/sweep problems. Thus, we deem the NES paradigm as one of the important abstractions to be layered on top of lower-level Grid services such as Globus [8], Legion [9], or Condor [10].

The Computational Grid as it was envisioned in [1] and as it is being standardized in the current Grid Forum activity [11] seems like an ideal platform for building, deploying, and using NES systems. It offers the considerable amount of compute and storage resources that are needed by large-scale scientific applications. Even though large network latencies might impact tightly coupled distributed applications, typical NES applications should be much more

*Computer Science and Engineering Department, University of California, San Diego, USA

†Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, Tokyo, Japan

‡Computer Science Department, University of Tennessee, Knoxville, Tennessee, USA

tolerant to latency. Fault-tolerance is an important issue in a complex federated environment such as the Grid. Fortunately, the simple NES programming model make it straightforward to implement simple recovery mechanisms. Finally, from a software stand-point the Grid infrastructure provides many basic mechanisms needed to effectively implement NES systems.

We do not advocate building a single Grid-enabled NES system. Indeed, there are several compelling reasons for allowing multiple NES systems to coexist on the Grid. First, different systems serve different user communities. A large part of the work involved in deploying a NES systems is to identify and package suitable scientific software to make them available as software modules. Given the sheer amount of available scientific software, some NES systems restrict themselves to particular domains and provides domain-specific UIs (e.g. NEOS [7]). It is not realistic to think that a single group or organization could try to meet the needs of all users. Second, different NES systems provide different user interfaces. This gives users and application developers more choice and it makes it possible for a small group of individual to develop an additional interface when needed (e.g. the Mathematica interface to NetSolve was developed by one individual outside the NetSolve group). Third, and most importantly, the groups that develop them generally use NES systems as vehicles for computer science research. For instance, an active research thrust is "scheduling". Each NES system implements different scheduling strategies in different contexts (minimum turn-around time, Grid economy model, with respect to storage constraints, etc.). It is vital to allow these research efforts to proceed freely within distinct but related research projects.

Nevertheless, all NES systems could benefit from a concerted effort that provides an intermediate *service layer* on top of the Grid. At this stage, the term "layer" means actual software, API specifications, wire protocol specifications, and concepts. The purpose of this layer is twofold. First, it will provide basic design concepts and mechanisms (built upon available Grid software infrastructure) that can be shared by NES systems on the Grid. This is to avoid duplication of effort and core development among NES systems and allow them to focus solely on the relevant research issues and on the end-user's needs. Second, having a common basis will allow NES systems to interoperate more easily. This is required in order to deploy a large number of software modules. For instance, if NetSolve contains software module specifications for a given set of general-purpose numerical routines, it should be possible for Ninf users to access these routines from Ninf, thereby benefiting from possible Ninf-specific features (e.g. specific UI, scheduling algorithms, automated interfaces to databases, etc.). One of the focuses of this paper is to define the functionalities and identify the software requirements for a common layer that can be used by NES systems to run on the Grid.

This paper is organized as follows. Section 2 lists and justifies current design choices in Ninf [2] and NetSolve [3]. Section 3 describes current application work based on NES systems. Section 4 identifies common services that should be layered on top of the Grid infrastructure for the implementation/deployment of NES systems and Section 5 concludes the paper.

2 NES Systems

2.1 Overview

One can distinguish five different fundamental components in an NES system:

- **Client:** provides multiple user interfaces and submit request for computations to servers.
- **Server:** receives requests from clients and executes required software modules on their behalf.
- **Database:** contains static and dynamic information about available computational resources (hardware and software).
- **Scheduler:** intercepts client requests and makes decision for mapping tasks to servers based on information located in the database.
- **Monitor:** dynamically monitors the status of computational resource and stores that information in the database.

These components can be implemented in various ways. In NetSolve for instance, the Scheduler and the Database are part of a single process (called the NetSolve *agent*), and the Monitors are embedded within the Servers. Some of these components can also be distributed in a distributed fashion (e.g. the Database). Based on the NetSolve and Ninf experiences, the following section highlights some basic design issues as well as current approaches in both projects.

2.2 Some Design Issues

2.2.1 RPC implementation

NES systems must provide an implementation of Remote Procedure Calls. In both Ninf's and NetSolve's first implementations the systems maintained a TCP connections between client and servers for the whole duration of each RPC. The main advantage of that approach is that server failure detection is straightforward on the client side. However, there are two major drawbacks. First, there is a scalability issue as the number of simultaneous connections from a single (client) process is limited by the O/S. We mentioned in Section 1 that an important class of target applications for NES systems on the computational Grid are scientific simulations and parameter searches. Typically, those applications can consist of tens to hundreds of thousands of tasks, raising the question of scalability. Second, and perhaps most importantly, maintaining connections assumes that the client is constantly on-line. This becomes less and less practical given that applications may run for days at a time, and that a number of users will probably use mobile resources to submit computation to the NES systems. Both Ninf and NetSolve, in their latest version, use protocols where no connection is maintained (*Connection-by-Necessity*). The client initially connects to a server to send input data and then disconnects. When the server completes, it tries to connect back to the client to send back output data. We believe that the added flexibility is well worth the added complexity of the protocol.

Both project also use a *client proxy*, that is a separate process that performs interactions between the client and the other components of the NES system as listed in the previous section. The use of a proxy adds flexibility in two ways. First, it makes it easy to write multiple user interfaces. Indeed, it suffices to define an interface-to-proxy protocol and all interfaces can re-use the same proxy. The expectation is that the interface-to-proxy protocol is much more simple than the protocol between the proxy and the rest of the NES components. Moreover, when design changes occur in the system, only the proxy's implementation needs to be upgraded. Second, using a proxy furthers interoperability as it is possible for one interface to use multiple proxies to use different types of resources. For instance, NetSolve's latest version provides two implementations of the proxy: one for interacting NetSolve servers; the other to interact with Globus resources. A few years ago, the Ninf and NetSolve teams had collaborated to develop a set of adapters [12] that made both systems interoperate. At the time, NetSolve did not use a client-proxy, and it became increasingly difficult to maintain the adapters. The proxy-based design would make it easy to convert the adapters into two new proxies: a NetSolve Ninf-proxy and a Ninf NetSolve-proxy. However, the purpose of this document is to advocate for even more advanced interoperability, as detailed in Section 4. Let us note that using a proxy leads to some overhead as all communication goes from the user interface to the proxy, and then onto other NES components. This is especially the case if all input/output application data were to transit through the proxy. NetSolve (and Ninf in its latest version) bypasses the proxy for application data transfers.

Another important part of the RPC implementation is the IDL that is used by both client and server to marshal input and output data/arguments and place actual calls to appropriate software modules. A main concern in NES systems is to keep the clients as *thin* as possible. In that sense, no IDL information is actually kept on the client, but instead downloaded from servers on demand. This is mandatory for viable NES deployment on the Grid. Indeed, one cannot expect users to upgrade their NES client each time a new IDL becomes available or is upgraded. The usage scenario is as follows. The NES system provides a way for users to find out what software modules are **currently** available and the syntax of RPC calls to those modules (order and types of arguments). For example, NetSolve provides a Web/CGI tool that users can query for information concerning all available software modules within a NetSolve system. That tool obtains real-time information from NetSolve agents and displays instructions for placing calls from various NetSolve client interfaces. The user then places the RPC call, the IDL is dynamically obtained by the NES client over the network and is used to marshal the arguments. Users do not need to actively download any software in order to be able to use a newly added software module on a server. The nature of the IDL itself is discussed in Section 4.2.1.

Finally, NES systems must define the wire-protocol that is going to be used among the different modules. NetSolve and Ninf used to employ binary formats for the protocol, making it lightweight and easy to parse. However, that also makes the protocol difficult to debug and extend, and well as precluded easy interoperability. By contrast, text-based formats are well-structures, easy to understand and extend, but are less efficient and require more of a parsing effort. There is a recent trend to use the XML

syntax as a de-facto standard in the Grid community, and we believe that it is a viable option for NES systems as well given the proliferation of XML-based tools and parsers. The expectation is that the overhead of using a more structured text-based protocol will be amortized over application data transfers. In this regard, NES developers must be careful to observe and possibly integrate with proposed Internet standards for XML-based RPCs, such as SOAP [13].

2.2.2 Deployment

Many issues pertain to the effective deployment of NES systems on the Grid. In this section we address four of these issues: security, information services, server management, and remote storage.

Security is by all means an important aspect of any widely deployed Grid system and authentication based on encryption is mandatory for any production-like usage. NetSolve provides Kerberos support with access-control lists on the server side. Ninf on the other hand has opted for an SSL-based authentication and authorization layer that is similar to the GSI in Globus. Let us describe the NetSolve implementation in more details. The latest NetSolve [14] version introduces the ability to generate access control lists, which are used to grant and deny access to the NetSolve servers. The NetSolve developers opted for using Kerberos V5 [15] services, as it is one of the most trusted and popular infrastructures for authentication services.

The server implements access control via a simple list of Kerberos principal names which usually consist of a name (often a UNIX username) and a realm, which defines the Kerberos "domain." This list of principals is maintained by the NetSolve server administrator and is kept in a private text file, which is consulted by the server. A request to a NetSolve server must be made on behalf of one of those principal names. If the principal name associated with the Kerberos credentials in the request appears in the list, and the credentials are otherwise valid, the request will be honored. Otherwise, the request will be denied.

A user first authenticates himself to Kerberos using a Kerberos utility like kinit. Through this utility, he talks to the Authentication Service on the Key Distribution Center (KDC) to get a Ticket Granting Ticket (TGT). This ticket is encrypted with the user's password. When the user wants to talk to a Kerberized NetSolve server, he uses the TGT to talk to the Ticket Granting Service (TGS) (which also runs on the KDC). The TGS verifies his identity using the TGT and issues a ticket for the NetSolve service. The user can then pass this service ticket to the Server who will verify his/her credentials and check to see if he/she is an authorized user. If the TGT is compromised, an attacker can only masquerade as a user until the ticket expires.

The NetSolve system supports the interoperation of Kerberized and non-Kerberized components. In either case the client sends a request to the server, and the established protocol dictates that, if required, the server must send an explicit request for authentication. At this point, the client can either abort the transaction (knowing it does not have the proper credentials) or attempt to authenticate itself to receive proper servicing. Currently there is no mechanism to allow the client to insist on authentication of the server - a Kerberized client will happily talk with either Kerberized or non-Kerberized servers. This Kerberized version of NetSolve performs no encryption of the data exchanged among NetSolve clients, servers, or agents, nor is there any integrity protection for the data stream.

Information services are used by users as well as by NES components to find out information about available resources and components. The Database introduced in Section 2.1 is an obvious place-holder for where the information must be stored, but there are many possibility for implementing the information services. NetSolve uses a custom protocol to store and retrieve information from that database. That protocol is used by NetSolve servers and agents as well as user-level tools (e.g. the Web/CGI tool mentioned in Section 2.2.1). The latest Ninf version uses a Database Manager that is in charge of interacting with possibly multiple databases. All database queries from Ninf Components go through the manager. At the moment, the Database in Ninf is LDAP-based and the database manager uses the LDAP protocol. Other protocols can be added when new databases of interest become available (e.g. Globus MDS). As Grid software infrastructure becomes widely deployed, it is most important to use existing Grid information services and the recent Ninf developments are going in that direction.

An issue related to information services is **server management**. NES servers provide access to software modules. The default Ninf and NetSolve strategy is to have these modules compiled and installed on a host before starting a server on that host. This approach, if simple to implement, is rather limiting and makes server management and software deployment difficult. A more flexible approach would be to allow servers to download pre-compiled soft-

ware modules on-the-fly from authorized software repositories or from other servers. Those modules could then be cached on the server for some limited amount of time until discarded. This is a very attractive approach from the user's perspective. As the demand for a given module increases, that module gets disseminated among an increasingly large number of servers. NetSolve provides a first attempt at a more flexible module management: precompiled software modules can be downloaded from the Netlib [16] repository and dynamically linked into the server. Security concerns are not taken into account because the assumption is that Netlib is an authoritative source of software. This "proof of concept" prototype performs only simplistic checks for binary and O/S compatibility that are sufficient for testing purposes. There are obvious difficulties in making such a model safe and effective in a production environment. However it would be relatively easy to build into the system an authentication mechanism to insure a user was getting an up to date version of what desired. It is expected that the Grid will provide standard ways to manage executables and match them to hosts.

Finally, as more real-world applications are ported to NES systems, it becomes increasingly clear that there is a need for **distributed/remote storage support**. One of the drawbacks of the pure RPC model is that input and output data keep being transferred back and forth between client and servers. We mentioned that both NetSolve and Ninf provide extensions that allow for some data-dependency analysis and thereby save on unnecessary data transfers. The underlying issue here is that current NES implementations do not provide ways of naming and storing application data in remote/distributed storage. Several efforts are aiming at providing infrastructure for Grid-wide distributed storage [17, 18]. Building on these efforts and given the added flexibility, it should be possible to implement the NES Scheduler so that it performs intelligent data movements. Our purpose here is not to design sophisticated scheduling algorithms, but rather to make sure that the required mechanisms for implementing such algorithms are available. Current work in progress within NetSolve aims at using IBP [17] for holding crucial application data (e.g. data that is shared among multiple tasks, or intermediary data that need not be returned to the user).

2.2.3 Interoperability

A major concern when implementing NES systems over the Grid is to ensure maximum interoperability with extent Grid services as well as among different NES systems. Interoperability with Grid services is already on its way with projects such as NetSolve and Ninf. Services under considerations are for instance those available in the Globus toolkit, but also other services such as the Network Weather Service [19]. Even though there have been some preliminary experiments (e.g. NetSolve/Ninf adapters [12]), interoperability among different NES systems is still far from being achieved. This is due to the fact that there has been little discussion or consensus concerning NES design issues. This document aims at taking a first step towards such a consensus in the context of the Grid. We have exposed some important design and deployment issues for NES systems. In the following section we describe NES applications and current experiences.

3 Current NES Applications

3.1 Scientific Simulations and Parameter-Space Searches

An important class of applications that lead themselves to RPC-style programming is that of scientific simulations, such as Monte-Carlo simulations, and parameter-space searches. Indeed, these applications are structured as a large number of independent tasks, that is with no task precedences. In what follows, we call these applications *Parameter Sweep Applications* or PSAs for short. Albeit their simple structure, deploying PSAs on the Computational Grid presents several challenges. In addition to the difficulties associated with deploying large-scale distributed applications, these applications often manipulate large amounts of data that must be shared/combined. This leads to a need for scheduling techniques that put an emphasis on co-locating data and computational tasks, and for a suitable distributed storage infrastructure in order to implement scheduling decisions. NES Systems have the opportunity to provide the most adequate mean of deploying PSAs on the Computational Grid.

PSAs arise in almost all fields of science and engineering [20, 21, 22, 23, 24, 25, 26, 27, 28] and we highlight here a few examples. MCell [20] is a **micro-physiology** application that uses 3-D Monte-Carlo simulation techniques to study molecular bio-chemical interactions within living cells.

MCell can be used to model neuro-transmition in the 3-D space between two cell membranes for different deformations of those membranes. INS2D [28] developed at NASA Ames Research Center is a **fluid dynamics application** that aims at solving the incompressible Navier-Stokes equations in two-dimensional generalized coordinates for both steady-state and time varying flow. NeuralObjects [29] in a **neuro-science** application which aims at simulating large-scale neuronal network models. It can be used to model a broad range of cellular population interactions from cellular automata, slime mold aggregation and heart tissue to visual cortex orientation selectivity, auditory nerve dynamics and complex thalamocortical brain circuitry. SNP [24] is a package for Semi-Non-Parametric time series analysis with many application in **economics** and general forecasting problems. TPHOT [22] is a **particle physics** application that simulates photon transport within high-density, high-temperature plasma.

3.2 A Case Study: APST

The AppLeS Parameter Sweep Template (APST) [30] project focuses on **scheduling** issues for PSAs in Grid environments. APST is a user-level middleware in the sense that it is not targeted on any particular application, but can be used by end-users to deploy their PSAs on the Grid. However, if it is to be usable, the APST software needs to perform application deployment as well as application scheduling. The goal of the APST implementors is then to deploy applications with minimum effort, and NES systems provide the appropriate mechanisms and level of abstraction. In its current version, APST can use NetSolve as a back-end and a Ninf back-end is being developed. APST by-passes the scheduling algorithm implemented in the NetSolve agent. The rationale behind this choice is two-fold. First, APST has access to very detailed application-level information that can be used for efficient scheduling. The current NetSolve API does not allow for that information to be conveyed to the NetSolve scheduler. Furthermore, the internal NetSolve scheduler does not implement the scheduling algorithms that are under investigation with APST and does not provide ways to "plug in" new scheduling algorithms. Second, APST aims at using multiple back-ends simultaneously so that it can access the largest possible number of resources available today. This is a compelling reason for letting APST do its own scheduling. The purpose of this section is not to provide a description of APST as all details on that software can be found in [30, 31]. However, we use the current APST design choices to highlight shortcomings of NES systems currently available. This will lead us to the next section in which we identify a set of NES-specific Grid services that address most of these shortcomings.

APST provides a Globus back-end. This may seem surprising since the programming model provided by the various Globus component APIs is much more complex than the NES programming model which has been identified in Section 1 as ideal for PSAs deployment. As a result, implementing APST on top of Globus required much more effort than on top of NetSolve. However, the reasons for providing a Globus back-end are compelling. Globus is widely deployed and maintained at many institutions and has been selected as the key Grid software infrastructure by key research centers (e.g. the NASA/IPG effort). NES systems such as NetSolve and Ninf have not reached this degree of deployment and commitment. We believe that this is due to the fact that there has not be any consensus or effort for standardizing the different NES projects. A goal of this paper is to set the bases for a discussion among NES developers that will further standardization and integration (see Section 4). Note that a Globus back-end to NetSolve is being developed. When available, APST will be able to use Globus resource via its NetSolve back-end.

APST does not provide a Ninf back-end at the moment. Once again, this is surprising since both NetSolve and Ninf are NES systems that share similar goals and general design decision. However, due to the lack of agreement between the IDL descriptions and the APIs of the two, the APST implementation on top of NetSolve cannot be easily converted to Ninf. Furthermore, APST implements ad-hoc mechanisms to allow NetSolve to use distributed storage software infrastructures (e.g. GASS [18] or IBP [17]). Using such infrastructures is necessary for good scheduling [32]. Note that the same mechanisms must be implemented separately for a Ninf back-end. The lack of standardized interaction between NES systems and distributed storage systems prevents easy implementation of APST over (multiple) NES systems. Lastly, the fact that each currently available NES system implements its own information services (different protocols and databases) makes it even more difficult to convert APST's NetSolve back-end to a Ninf back-end. Different mechanisms need to be used to access resource information. Using agreed-upon Grid information services for NES implementations (e.g. the Globus

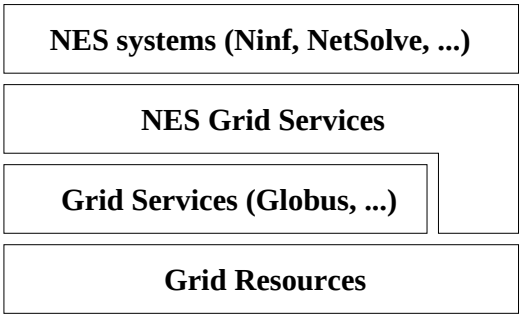

Figure 1: Software hierarchy

MDS with the LDAP protocol) would address this problem. Nevertheless, a Ninf back-end is currently being developed for APST in order to access Ninf resources available today.

It now appears clearly that if NES systems are to provide a viable way to access Grid resources, there needs to be agreement on common core functionalities, some of which build directly on the available Grid infrastructure. The set of NES Grid services suggested in the following section are a first step towards that consensus.

4 NES Grid Services

4.1 Scope

Our purpose is not to provide the specification for a Grid-enabled NES system but rather to identify common middle-level services (at the level of or above basic Grid services) that can be used effectively to build, deploy, and use NES systems in the Grid context. Figure 1 depicts the resulting software hierarchy.

The following section proposes and discusses an initial set of Grid services for NES systems. Many of these services and their possible implementations are inspired by some of the recent developments in NetSolve and Ninf described in Section 2.2. Note that we use the term *service* rather loosely and a better term might be *building block*. Indeed, some of our "services" do not entail actual implementations, but rather consensus on a standard (e.g. IDL scheme).

4.2 Services

4.2.1 IDL

Designing an appropriate IDL for NES systems has proven to be quite a difficult task. Difficulties arise for two reasons. First, the target applications come from different domains and are really diverse in terms of implementation. Second those applications use legacy code that often contains many idiosyncrasies. The IDL must then be general enough to be amenable to many different applications, while allowing for very specific and detailed features to accommodate legacy code. NetSolve and Ninf approaches at defining an IDL are different and both effective in their own way. Ninf's IDL scheme makes it easy to write interface descriptions (they look like enhanced C function prototypes). However, it lacks features to accommodate a number of behaviors of legacy code. By comparison, NetSolve's IDL scheme is lower-level and less elegant, but is more flexible. In fact, part of the interface description can be a actual fragment of C code with special IDL-related macros. Although CORBA does already define a industry-standard IDL for distributed computing, and that there have been several work that attempted to exploit CORBA for Grid efforts, CORBA IDL largely is designed for wrapping business applications, and a number of otherwise necessary features for high-performance scientific computing [33] are not available.

Agreeing on and specifying an IDL for NES systems on the Grid is key to making such systems truly usable. The experience already gained by the NetSolve and the Ninf team should prove valuable to achieve that goal.

4.2.2 Protocol

The goal of the NetSolve/Ninf adapters that we mentioned in Section 2.2.3 was to perform protocol translation between the two systems. That translation was needed because of a lack of consensus regarding which protocol should be used, and it was difficult because both projects used binary-based protocols. The latest Ninf version moved to a text-based protocol that uses the XML syntax and it seems like a good first step towards setting the bases of a common NES protocol on the Grid. The next version of NetSolve will also have this attractive feature. The challenge will be to keep core functionalities of the protocol general while allowing customizations for different NES systems. The expectation is that the use of XML will make it possible to implement those customization in a parsable form and will therefore make what protocol-translation is needed (for interoperability) straightforward.

4.2.3 Information and User Services

The Grid aims at providing a general information service (e.g. the Globus MDS) that can be used to store information. It is clear that the NES Database (see Section 2.1) should leverage that service to store and retrieve information concerning hardware and software resources. Part of that information will be used by NES components (e.g. the Scheduler) whereas part of it should be viewable/searchable by the NES user. The current trend for managing distributed information on the Grid is to use LDAP (the latest Ninf version uses it). Like for the protocol described in the previous section, the goal is to define a common core for the kind of NES-related information that needs to be stored in the information system (to further interoperability), while allowing for customizations for different NES systems.

There are also a number of efforts for making non-programmable interfaces, typically GUI/Web based, to the Grid. These are now coined as "scientific portals" or "Grid portals". For NES systems, there is work to present the users with what are effectively portal front-ends to access the servers on the back-end. A good example is the Punch system [5], which defines a web-based access framework to canned packages, and has been in active use. Nimrod, NetSolve, and Ninf already provides some form of web-based access to their respective infrastructure. More recent work by the Ninf group attempts to unify the portals technology amongst different NES as well as other Grid systems based on Java Jini technology [34].

Still, there is little consensus on what kind of standard GUI interface should be provided to the user, what kind/type of information under what format, what are the protocols/APIs for accessing the information residing within a NES system, etc. Standardizing on at least some of these, in accordance with the standards set by Grid Forum working groups, would be a strong step in making NES services widely accessible over the Grid.

4.2.4 Server Management

NES systems would benefit from using a common scheme for managing the software modules whose interfaces are described with the IDL. The hope is to use deployed executable management Grid services to identify and match software modules with candidate hosts. Such a mechanism would make it possible to perform software/hardware binding on demand for incoming client requests as described in Section 2.2.2. The first step is then to specify a formal definition of a software module (IDL + binaries) and to provide a simple API that manipulates those modules (for searching, downloading, caching, etc.).

4.2.5 Distributed Storage

As explained in Section 2.2.2, interacting with Grid distributed storage systems is a way to improve performance for applications for which input/output data transfers are critical. Even though it may be that NES developers can directly use Grid storage services, a more general scheme is needed. Indeed, the use of remote storage is not imposed but optional. Furthermore, multiple storage systems can be used by a single user (e.g. GASS, Web server, local disk, local NFS). The idea is to provide an additional level of abstraction for naming and locating data stored on disk under diverse Grid storage systems. That abstraction must be supported by the IDL and it will be the responsibility of the NES developer to retrieve and store data according to IDL descriptions. The work in [30] is taking steps in that direction as an adequate distributed storage system is vital to efficient scheduling of certain applications (see Section 3.2).

5 Conclusion

In this paper we have made a case for the Network-Enable Server (NES) paradigm as a promising way to efficiently deploy large-scale scientific applications on the Computational Grid. Based on the experience gained while developing the NetSolve and Ninf systems, we have identified relevant design issues that must be addressed to deploy NES systems. Taking the example of the APST software, we made a case for better interoperability between NES systems and the Grid infrastructure as well as among each other. To this end, we proposed a set of core *services* that provide a middle layer on top of the Grid infrastructure. Realistically, many specific features of NES systems (high-level user interfaces, scheduling strategies, policies, etc.) are not specified as core services. This paper sets the bases for the standardization efforts that must take place among NES systems if they are to be used in the Computational Grid context successfully.

References

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

[2] S. Sekiguchi, M. Sato, H. Nakada, S. Matsuoka, and U. Nagashima. Ninf : Network based Information Library for Globally High Performance Computing. In *Proc. of Parallel Object-Oriented Methods and Applications (POOMA), Santa Fe*, pages 39–48, February 1996.

[3] 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. Also in Proceedings of Supercomputing'96, Pittsburgh.

[4] 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.

[5] N.H. Kapadia, J.A.B. Forter, and C.E. Brodley. Predictive Application-Performance Modeling in a Computational Grid Environment. In *Processings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC8)*, 1999.

[6] P. Arbenz, W. Gander, and M. Oettli. The Remote Computational System. *Parallel Computing*, 23(10):1421–1428, 1997.

[7] J. Czyzyk, M. Mesnier, and J. Moré. NEOS: The Network-Enabled Optimization System. Technical Report MCS-P615-1096, Mathematics and Computer Science Division, Argonne National Laboratory, 1996.

[8] I. Foster and K Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[9] A. Grimshaw, A. Ferrari, F.C. Knabe, and M. Humphrey. Wide-area computing: Resource sharing on a large scale. In *IEEE Computer 32(5)*, volume 32(5), May 1999. page 29-37.

[10] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceesings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.

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

[12] http://ninf.etl.go.jp/ninf-netsolve.html.

[13] http://www.w3.org/TR/SOAP/.

[14] D. Arnold, S. Browne, J. Dongarra, G. Fagg, and Keith Moore. Secure Remote Access to Numerical Software and Computational Hardware. In *Proceesings DoD HPC Users Group Conference (HPCUG 2000)*, June 2000.

[15] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, September 1994.

[16] S. Browne, J. Dongarra, E. Grosse, and T. Rowan. The Netlib Mathematical Software Repository. *D-Lib Magazine*, Sep. 1995. Accessible at http://www.dlib.org/.

[17] 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.

[18] 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.

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

[20] J.R. Stiles, T.M. Bartol, E.E. Salpeter, , and M.M. Salpeter. Monte Carlo simulation of neuromuscular transmitter release using MCell, a general simulator of cellular physiological processes. *Computational Neuroscience*, pages 279–284, 1998.

[21] 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.

[22] A. Majumdar. Parallel Performance Study of Monte-Carlo Photon Transport Code on Shared-, Distributed-, and Distributed-Shared-Memory Architectures. In *Proceedins of the 14th Parallel and Distributed Processing Symposium, IPDPS'00*, pages 93–99, May 2000.

[23] 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.

[24] A. Ronald Gallant and G. Tauchen. SNP: A Program for Nonparametric Time Series Analysis. Duke economics Working Paper #95-26, Duke University, 1997. v8.6 (revised 1997).

[25] W.R. Nelson, H. Hirayama, and D.W.O. Rogers. The EGS4 Code system. Technical Report SLAC-265, Stanford Linear Accelerator Center, 1985.

[26] S.J. Sciutto. AIRES users guide and reference manual, version 2.0.0. Technical Report GAP-99-020, Auger project, 1999.

[27] A. Amsden, J. Ramshaw, P. O'Rourke, and J. Dukiwica. Kiva: A computer program for 2- and 3-dimensional fluid flows with chemical reactions and fuel sprays. Technical Report LA-10245-MS, Los Alamos National Laboratory, 1985.

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

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

[30] 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.

[31] http://apples.ucsd.edu/apst.

[32] 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.

[33] T. Suzumura, T. Nakagawa, S. Matsuoka, H. Nakada, and S. Sekiguchi. Are Global Computing Systems Useful? Comparison of Client-server Global Computing Systems Ninf, NetSolve Versus CORBA. In *Proceedins of the 14th Parallel and Distributed Processing Symposium, IPDPS'00*, pages 547–559, May 2000.

[34] http://www.sun.com/jini.