HAL
archives-ouvertes.fr

Adding Data Persistence and Redistribution to NetSolve

Frédéric Desprez, E. Jeannot

▶ **To cite this version:**

Frédéric Desprez, E. Jeannot. Adding Data Persistence and Redistribution to NetSolve. [Research Report] LIP RR-2001-39, Laboratoire de l'informatique du parallélisme. 2001, 2+9p. hal-02102008

HAL Id: hal-02102008

https://hal-lara.archives-ouvertes.fr/hal-02102008

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Laboratoire de l'Informatique du Parallélisme

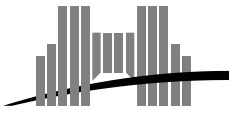
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

Adding Data Persistence and Redistribution to NetSolve

F. Desprez
E. Jeannot

Décembre 2001

Research Report N° 2001-39



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Adding Data Persistence and Redistribution to NetSolve

F. Desprez
E. Jeannot

Décembre 2001

Abstract

The implementation of Network Enabled Servers (NES) on grid environments requires to lower the cost of communications. NetSolve, a NES environment developed at University of Tennessee Knoxville, sends data back to the client at the end of every computation. This implies unneeded communications when computed data are needed by an other server in further computations. In this paper, we present the modifications we made to the NetSolve protocol in order to overcome this drawback. We have developed a set of new functions and data structures that allow users to order servers to keep data in place and to redistribute them directly to an other server when needed.

Keywords: Calcul distribuée client-serveur, serveurs accessibles via the réseau, NetSolve, persistance de données, redistribution de données.

Résumé

L'implémentation de serveurs accessibles via the réseau (NES) sur des environnements de type grille requiert une réduction du coût des communications. NetSolve, un environnement de type NES développé à l'Université du Tennessee Knoxville, renvoie les données vers le client à la fin de chaque calcul. Ceci ajoute des communications inutiles lorsque les données calculées doivent être réutilisées dans des calculs futurs. Dans cet article, nous présentons les modifications que nous avons apporté au protocole de NetSolve pour supprimer cet inconvénient. Nous avons développé un ensemble de nouvelles fonctions et de structures de données qui permettent aux serveurs de conserver les données sur place et de les redistribuer directement vers un autre serveur lorsque c'est nécessaire.

Mots-clés: client-server distributed computing, Network Enabled Servers, NetSolve, data persistence, data redistribution.

Adding Data Persistence and Redistribution to NetSolve*

F. Desprez†, E. Jeannot‡,§

December, 2001

1 Introduction

Due to the progress in networking, computing intensive problems in several areas can now be solved using networked scientific computing. In the same way that the World Wide Web has changed the way that we think about information, we can easily imagine the types of applications we might construct if we had instantaneous access to a supercomputer from our desktop. The RPC approach [7, 8] is a good candidate to build Problem Solving Environments on the Grid. Several tools exist that provide this functionality like Netsolve [5], NINF [10], NEOS [9], or RCS [1]. However, none of them do implement data persistence in servers and data redistribution between servers. This means that once a server has finished its computation, output data are immediately sent back to the client and input data are destroyed. Hence, if one of these data is needed for another computation, the client has to bring it back again on the server. This problem as been partialy tackled in NetSolve with the new request sequencing feature [2]. However, the current request sequencing implementation does not allow to handle multiple servers. In this paper, we show how we have modified the NetSolve protocol in order to implement data persistence and data redistribution between servers.

The remaining of this paper is organized as follows. The NetSolve architecture and the objectives of this work are described in Section 2. In Section 3.1, we describe the modifications made to the server. Section 3.2 is dedicated to a new data structure that tells how to redistribute objects from a server to an other server. In Section 3.3, we describe all the NetSolve functions involved in data redistribution and data persistence operations. In Section 3.4, the modifications done to NetSolve's scheduler are described and a program example is presented in Section 4. Finally and before some concluding remarks, we describe in Section 5 our experimental results.

2 Background

2.1 NetSolve

Overview NetSolve is a NES environment for executing computations on remote servers. Its architecture is based on the client-agent-server model which is composed of three components:

*This work has been supported in part by the ARC INRIA OURAGAN. URL: http://www.ens-lyon.fr/~desprez/OURAGAN/

†ReMaP, LIP-ENS Lyon. INRIA Rhône-Alpes, 46, allée d'Italie, 69364 Lyon cedex 07, France.

‡Résédas, LORIA. 615, rue du Jardin Botanique, BP 101, 54602 Villers-Les-Nancy, cedex, France.

§Part of this work has been done while the author was in postdoc at LaBRI.

- the *agent* is the manager of the architecture. It knows the state of the system. Its main role is to find servers that will be able to solve as efficiently as possible client requests,
- *servers* are computational resources. Each server registers to an agent and then waits for client requests. Computational capabilities of a server are known as *problems* (matrix multiplication, sort, linear systems solving, ...). A server can be sequential (executing sequential routines) or parallel (executing operations in parallel on several nodes),
- a *client* is a program that requests for computational resources. It asks the agent to find a set of servers that will be able to solve its problem. Data transmitted between a client and a server are called *objects*. Thus, an *input* object is a parameter of a problem and an *output* object is a result of a problem.

The architecture works as follows. First, an agent is launched. Then, servers register to the agent by sending a list of problems they are able to solve as well as the speed and the workload of the machine on which they are running and the network's speed (latency and bandwidth) between the server and the agent. A client asks the agent to solve a problem. The agent scheduler selects a set of servers that are able to solve this problem and sends back the list to the client. The client sends the input objects to one of the servers. The server performs the computations and returns the output objects to the client. Finally local server objects are destroyed.

Request Sequencing One of the new feature proposed since NetSolve 1.3 is the *request sequencing* [2]. Request sequencing consists in scheduling a sequence of NetSolve calls on one server. This is a high level functionality since only two new sequence delimiters `netsl_sequence_begin` and `netsl_sequence_start` are added in the client API. The calls between those delimiters are evaluated at the same time and the data movements due to dependencies are optimized. However request sequencing has the following deficiencies. First it does not handle multiple servers because no redistribution is possible between servers. An overhead is added for scheduling NetSolve requests. `for` loops are forbidden within sequences, and finally the execution graph must be known at compile time and cannot depend on results computed within the sequence.

2.2 Goal of our Work

First, we should allow servers to keep objects in place to be able to use these objects again in a new problem without sending them back and forth to and from the client. Secondly, we want to enable inter-server communications to allow data redistribution between servers. We also want to keep a backward compatibility. Data persistence and data redistribution require the client API to be modified but we want standard clients to continue to execute normally. Moreover, we want our modifications to be standalone. This means that we do not want to use an other software to implement our optimizations. Hence, NetSolve users do not have to download and compile new tools. Finally, we wanted our implementation to be very flexible without the restrictions imposed by NetSolve's request sequencing feature.

3 Modifications Done to NetSolve

3.1 Server Modifications

NetSolve communications are implemented using sockets. In this section, we give details about the low level protocols that enable data persistence and data redistribution between servers.

3.1.1 Data Persistence

When a server has finished its computations, it keeps all the objects locally, listen to a socket and waits for new orders from the client. So far, the server can receive five different orders.

1. *Exit*. When this order is received, the server terminates the transaction with the client, exits, and therefore data are lost. Saying that the server exits is not completely correct. Indeed, when a problem is solved by a server, a process is forked, and the computations are performed by the forked process. Data persistence is also done by the forked process. In the following, when we say that the server is terminated, it means that the forked process exits. The NetSolve server is still running and it can solve new problems.
2. *Send one input object*. The server must send an input object to the client or to an other server. Once this order is executed, data are not lost and the server is waiting for new orders.
3. *Send one output object*. This order works the same way than the previous one but a result is sent.
4. *Send all input objects*. It is the same as “send one input object" but all the input objects are sent.
5. *Send all output objects*. It is the same as “send one output object" but all the results are sent.

3.1.2 Data Redistribution

When a server has to solve a new problem, it has first to receive a set of input objects. These objects can be received from the client or from an other server. Before an input object is received, the client tells the server if this object will come from a server or from the client. If the object comes from the client, the server has just to receive the object. However, if the object comes from an other server, a new protocol is needed. Let call S_1 the server that has to send the data, S_2 the server that is waiting for the data, and C the client.

1. S_2 establishes a socket s on an available port p.
2. S_2 sends this port to C.
3. S_2 waits for the object on socket s.
4. C orders S_1 to send one object (input or output). It sends the object number, forward the number of the port p to S_1 and sends the hostname of S_2.
5. S_1 connects a socket to the port p of S_2.
6. S_1 sends the object directly to S_2 on this socket: data do not go through the client.

3.2 Client Modifications

3.2.1 New structure for the client API

When a client needs a data to stay on a server, three informations are needed to identify this data. (1) Is this an input or an output object? (2) On which server can it be currently found? (3) What is the number of this object on the server?

We have implemented the structure `ObjectLocation` to describe these needed informations. `ObjectLocation` has 3 fields:

1. `request_id` which is the request number of the non-blocking call that involves the requested data. If `request_id` equals -1, this means that the data is available on the client.
2. `type` can have two values: `INPUT_OBJECT` or `OUTPUT_OBJECT`. It describes if the requested object is an input object or a result.
3. `object_number` is the number of the object as described in the problem descriptor.

3.2.2 Modification of the NetSolve code

When a client asks for a problem to be solved, an array of `ObjectLocation` data structures is tested. If this array is not `NULL`, this means that some data redistribution have to be issued. Each element of the array corresponds to an input object. For each input object of the problem, we check the `request_id` field. If it is smaller than 0, no redistribution is issued, everything works like in the standard version of Netsolve. If the `request_id` field is greater than or equal to zero then data redistribution is issued between the server corresponding to this request (it must have the data), and the server that have to solve the new problem.

3.3 Set of New Functions

In this section, we present the modifications of the client API that uses the low-level server protocol modifications described above. These new features are backward compatible with the old version. This means that an old NetSolve client will have the same behavior with this enhanced version: all the old functions have the same semantic, except that when doing a non-blocking call, data stay on the server until a command that terminates the server is issued. These functions have been implemented for both C and Fortran clients. These functions are very general and can handle various situations. Hence, unlike request sequencing, no restriction is imposed to the input program.

3.3.1 Wait Functions

We have modified or implemented three functions: `netslwt`, `netslwtcnt` and `netslwtnr`. These functions block until computations are finished. With `netslwt`, the data are retrieved and the server exits. With `netslwtcnt` and `netslwtnr`, the server does not terminate and other data redistribution orders can be issued. The difference between these two functions is that unlike `netslwtcnt`, `netslwtnr` does not retrieve the data.

3.3.2 Terminating a Server

The `netslterm` orders the server to exit. The server must have finished its computation, local object are then lost.

3.3.3 Probing Servers

As in the standard NetSolve, `netslpr` probes the server. If the server has finished its computations, results are not retrieved and data redistribution orders can be issued.

3.3.4 Retrieving Data

A data can be retrieved with the `netslretrieve` function. Parameters of this functions are the type of the object (input or output), the request, the object number and a pointer where to store the data.

3.3.5 Redistribution Function

`netslnbdist`, is the function that performs the data redistribution. It works like the standard non-blocking call `netslnb` with one more parameter: an object location array, that describes which objects are redistributed and where they can be found.

3.4 Agent Scheduler Modifications

We have modified the agent's scheduler to take into account the new data persistence persistence. The standard scheduler assumes that all data are located on the client. Hence, communication costs do not depend on the fact that a data can already be distributed. We have modified the agent's scheduler and the protocol between the agent and the client in the following way. When a client asks the agent for a server, it also sends the location of the data. Hence, when the agent computes the communication cost of a request for a given server, this cost can be reduced by the fraction of data already hold by the server. It also uses these informations for scheduling purpose.

4 Code Example

We show a code that illustrates the features described in this paper. It executes 3 matrix multiplications: `c=a*b`, `d=e*f`, and `g=d*a`, where `a` is redistributed from the first server and `d` is redistributed from the second one. We will suppose that matrices are correctly initialized and allocated. In order to simplify this example we will suppose that each matrix has `n` rows and `n` columns and test of requests are not shown.

```
netslmajor("Row");
trans='N';
alpha=1;
beta=0;

/* c=a*b */
request_c=netslnb("dgemm()",&trans,&trans,n,n,n,&alpha,a,n,b,n,&beta,c,n);

/* d=e*f */
request_d=netslnb("dgemm()",&trans,&trans,n,n,n,&alpha,e,n,f,n,&beta,d,n);

/* COMPUTING REDISTRIBUTION */

/* 7 input object for dgemm */

nb_objects=7;
redist=(ObjectLocation*)malloc(nb.objects*sizeof(ObjectLocation));

/* All objects are first supposed to be hosted on the client */
```

```
for(i=0;i<nb_object;i++)
  redist[i].request.id=-1;

/* We want to compute g=d*a */

/* d is the input object No 3 of dgemm and the output object No 0 of request_d */

redist[3].request_id=request_d;
redist[3].type=OUTPUT_OBJECT;
redist[3].object_number=0;

/* a is the input object No 4 of dgemm and the input object No 3 of request_c */

redist[4].request_id=request_c;
redist[4].type=INPUT_OBJECT;
redist[4].object_number=3;

request_g=netslnbdist("dgemm()",redist,&trans,&trans,n,n,n,&alpha,
                      NULL,n,NULL,n,&beta,g,n);
```

5 Experiments

Scilab is a tool heavily used in the mathematic community [6]. As Matlab, it allows to execute scripts for engineering and scientific computations. However, it has some limitations since it is not parallelized. The goal of Scilab$_{//}$ [4], developed in the OURAGAN project[1]is to allow an efficient and transparent execution of Scilab in a grid environment. Various approaches have been implemented in order to meet these objectives. One of these is to execute Scilab computations on dedicated servers distributed over the Internet. In order to achieve this goal, we chose to use NetSolve [3, 5] as a middleware between the Scilab console and our computational servers.

Figures 2 and 3 show our experimental results using NetSolve as a NES environment for solving matrix multiplication problems in a grid environment.

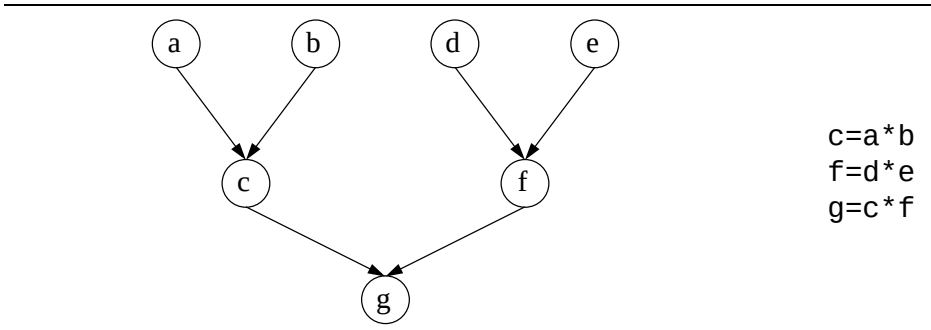


Figure 1: Matrix multiplications program task graph.

In Figure 2, we ran a NetSolve client that performs 3 matrix multiplications using 2 servers. The client, agent, and servers are in the same LAN and are connected through Ethernet. Computations and task graphs are shown in Figure 1. The first two matrix multiplications are independent and can be done in parallel on two different servers. We see that the time taken by Scilab is about

[1] http://www.ens-lyon.fr/~desprez/OURAGAN

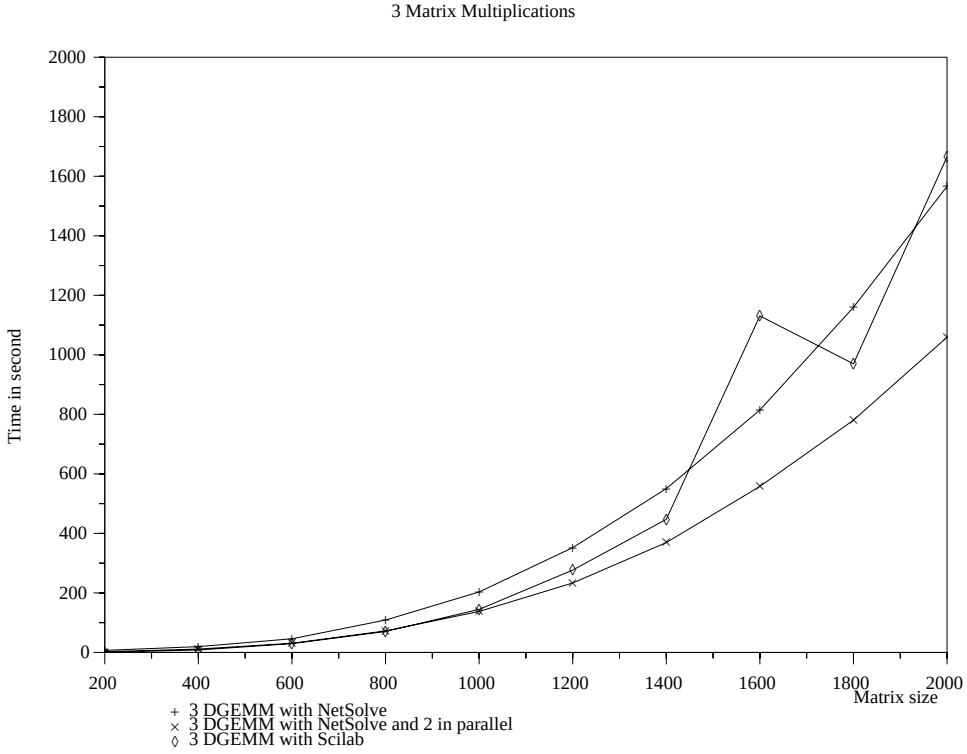
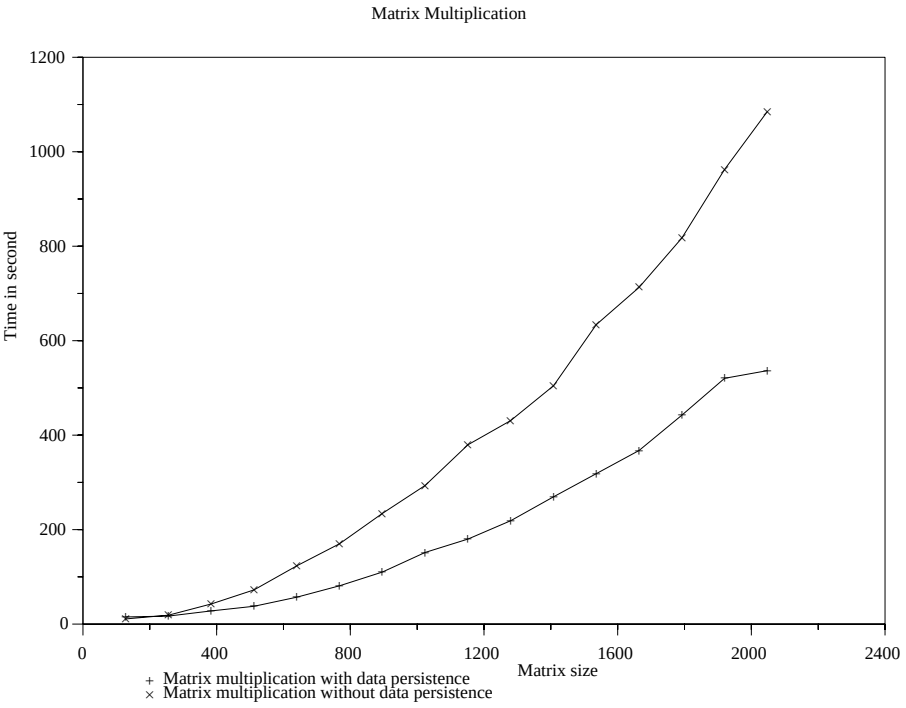


Figure 2: Matrix multiplications using NetSolve on a cluster of PCs.

the same than the time taken using NetSolve when sequentializing the three matrix multiplications. When doing the first two ones in parallel on two servers using the redistribution feature, we see that we gain exactly one third of the time, which is the best possible gain. These results show that NetSolve is very efficient in distributing matrices in a LAN and that non-blocking calls to servers are helpful for exploiting coarse grain parallelism.

Then, we have performed a matrix multiplication (Figure 3). The client and agent were located in one University (Bordeaux) but servers were running on the nodes of a cluster located in Grenoble[2]. The computation decomposition done by the client is shown in Figure 3. Each matrix is decomposed in 4 blocks, each block of matrix A is multiplied by a block of matrix B and contributes to a block of matrix C. The first two matrix multiplications were performed in parallel. Then, input data were redistributed to perform matrix multiplications 3 and 4. The last 4 matrix multiplications and additions can be executed using one call to the level 3 BLAS routine `DGEMM` and requires input and output objects to be redistributed. Hence, this experiment uses all the features we have developed. We see that with data persistence (input data and output data are redistributed between the servers and do not go back to the client), the time taken to perform the computation is more than twice faster than the time taken to perform the computation without data persistence (in that case, the blocks of A, B, and C are sent back and forth to the client). This experiment demonstrates how useful the data persistence and redistribution features that we have implemented within NetSolve are.

[2]Grenoble and Bordeaux are two French cities separated by about 500 miles.



1. $C_{11} = A_{11}B_{11}$
2. $C_{22} = A_{21}B_{12}$
3. $C_{12} = A_{11}B_{12}$
4. $C_{21} = A_{21}B_{11}$
5. $C_{11} = C_{11} + A_{12}B_{21}$
6. $C_{22} = C_{22} + A_{22}B_{22}$
7. $C_{12} = C_{12} + A_{12}B_{22}$
8. $C_{21} = C_{21} + A_{22}B_{21}$

Figure 3: Matrix multiplication using block decomposition.

6 Conclusion and future work

In this paper, we have presented how we added data persistence and data redistribution to NetSolve. Our contributions are the following. We have modified the server in order to be able to keep data in place on servers after computation. The server waits for orders from the client and is able to redistribute data to an other server when needed. We have modified the client API in order to simply write client programs involving data persistence and data redistribution. Then, we have modified the agent's scheduler. Allocation decisions take into account the fact that some data may already be distributed. These modifications keep the backward compatibility with old Netsolve clients. Finally, these features are standalone and there is no need to download or compile an other software.

Future work are directed towards the following directions. First, we want to implement new functionalities such as deleting data on a server. Second, we need to enhance the scheduler to be able to take more accurate decisions. Third, we are aware that using data persistence and data redistribution requires to rewrite NetSolve client programs. We would like to simplify the client API in order to increase the transparency of the proposed features. Finally, we want to implement data redistribution and data persistence for parallel servers. This last development requires the development of a data redistribution routine between servers that will be able to transfer huge distributed data sets.

References

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

[2] Dorian C. Arnold, Dieter Bachmann, and Jack Dongarra. Request Sequencing: Optimizing Communication for the Grid. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, *Euro-Par 2000 Parallel Processing, 6th International Euro-Par Conference*, volume 1900 of *Lecture Notes in Computer Science*, pages 1213–1222, Munich Germany, August 2000. Springer Verlag.

[3] Dorian C. Arnold and Jack Dongarra. The NetSolve Environment: Progressing Towards the Seamless Grid. In *International Conference on Parallel Processing (ICPP-2000)*, Toronto Canada, August 2000.

[4] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab$_{//}$, the OURAGAN Project. *To appear in Parallel Computing*, 2001.

[5] H. Casanova and J. Dongarra. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212 – 213, Fall 1997.

[6] Claude Gomez, editor. *Engineering and Scientific Computing with Scilab*. Birkhäuser, 1999.

[7] S. Matsuoka and H. Casanova. Network-Enabled Server Systems and the Computational Grid. `http://www.eece.unm.edu/~dbader/grid/WhitePapers/GF4-WG3-NES-whitepaper%-draft-000705.pdf`, July 2000. Grid Forum, Advanced Programming Models Working Group whitepaper (draft).

[8] S. Matsuoka, H. Nakada, M. Sato, , and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. `http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf`, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[9] NEOS. `http://www-neos.mcs.anl.gov/`.

[10] NINF. `http://ninf.etl.go.jp/`.