SmartNetSolve: High-Level Programming System for High Performance Grid Computing

Thomas Brady[1], Eugene Konstantinov[2], Alexey Lastovetsky[1]

[1]School of Computer Science and Informatics
University College Dublin,
Belfield, Dublin 4, Ireland
{thomas.brady, alexey.lastovetsky}@ucd.ie

[2]IBM Ireland
IDA Business Park, Ballycoolin Industrial Estate,
Blanchardstown, Dublin, Ireland
ekonstan@ie.ibm.com

Abstract

The paper presents SmartNetSolve, an extension of NetSolve, the programming system for high performance Grid computing. The extension is aimed at higher performance of Grid applications by improving the mapping of remote tasks and allowing them to communicate directly. To achieve more optimal mapping SmartNetSolve allows a group of tasks to be scheduled collectively, meanwhile NetSolve only allows for individual and independent mapping of remote tasks. SmartNetSolve also extends the communication model of the application by allowing remote tasks to communicate directly. The paper presents the overall design of the SmartNetSolve programming system with particular focus on its motivation and the underlying execution and communication models.

1. Introduction

NetSolve [1] is a programming system for high performance distributed computing on global networks using a remote procedure call mechanism. It deals with the situation when some computation tasks of the application can only be performed on remote computers. A NetSolve client program includes calls to the NetSolve client interface, each specifying the computation task and locations of the input and output data of the task on the client computer. When the program runs, the NetSolve programming system selects the remote computer to perform the task, transfers input data from the client computer to the server one, and delivers output data from the server computer to the client one.

The mapping of the tasks to remote computers is the core operation having a major impact on the performance of the application. In NetSolve, each task is mapped individually and independently of other tasks of the application. Thus, the NetSolve agent responsible for the mapping deals with a sequence of independent tasks, each of which should be mapped to one of the remote computers that are able to perform the task.

The client-sever NetSolve programming model is simple and easy to understand and use. It is also easy to implement. It supports fault tolerance in a natural way. At the same time, the model imposes a number of restrictions, which are able to significantly limit the performance of the application. First of all, the model does not allow remote severs to communicate directly. This enforces bridge communications and significantly limits the space of possible "virtual" communication links between computers.

Secondly, the model does not allow a group of related tasks, executing a logical unit of the application, to be mapped together. The model supports minimization of the execution time of each individual task of the application rather than minimization of the execution time of the whole application. Collective mapping of the group, which takes into account relationships between the tasks (such as data dependencies and the order of execution), could result in more optimal mapping of the tasks and hence in faster execution of the application as a whole. Unlike the individual mapping, the collective mapping can also take advantage of direct communications between remote servers by considering a number of possible communication schemes interconnecting the tasks when mapping.

Indeed, the communication model of NetSolve results in a communication network which has a *star* topology as illustrated in Figure 1. Therefore, in this case, for any given mapping of a group of tasks to remote servers there will be only one communication path between any pair of servers that should be considered when mapping. This path consists of two communication links connecting the servers with the client machine. Any other path connecting the two servers obviously results in a higher communication cost.

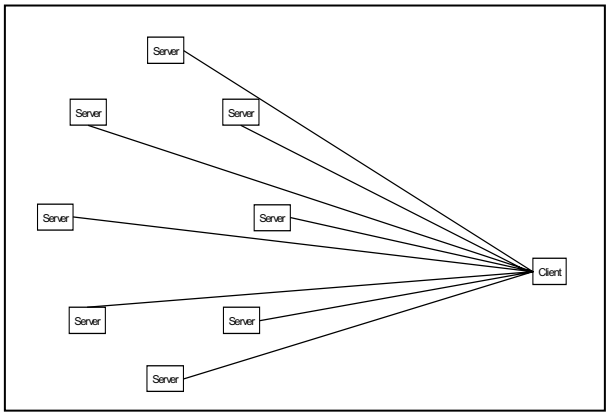

Figure 1. Communication model of NetSolve

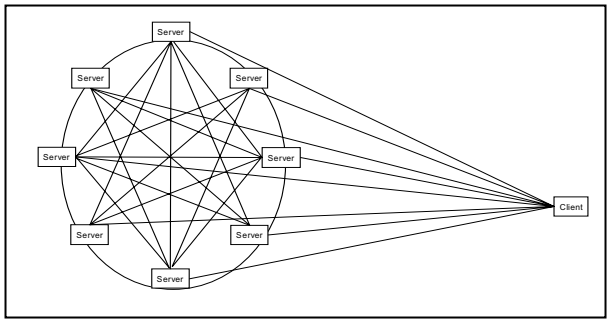

Figure 2. Communication model of SmartNetSolve

However, if the communication network is fully connected, then there will be multiple independent paths connecting the servers and each of these paths can be considered when mapping. In other words, for each mapping of a group of tasks to remote servers in a star communication network there is only one fixed communication scheme that can be employed. However, when a group of tasks are mapped on a fully connected network there are many communication schemes to choose from. Obviously, individual mappings cannot take advantage of the communication links of the fully connected network as the characteristics of a group of tasks such as virtual links between tasks are not taken into account when mapping.

The mapping of a group of tasks on a fully connected network not only involves the mapping of tasks to servers but also the mapping of the virtual links between tasks (i.e. links representing data dependencies) on to the communication paths of the network. This increases the mapping solution space and allows for further optimization to be achieved by choosing the optimal paths for data to traverse between servers.

In order to support this collective mapping of a group of tasks on the fully connected network of servers, we extend the NetSolve execution and communication model as follows:

- The execution model of NetSolve is extended so that mapping of a task can be separated from its execution, thus supporting the mapping of a group of tasks.
- The communication model of NetSolve is extended. to allow direct data transfers between remote servers

The NetSolve performance models which are used by the agent when mapping tasks are also extended to support collective mapping of a group of tasks. NetSolve currently maps an individual task on to the network based on the performance models that specify:

- The star network,
- The individual task.

The performance models of NetSolve are extended to specify:

- The fully connected network,
- A group of tasks.

The paper is structured as follows. In section 2, we outline the extended performance models described above. In section 3, we outline the extended execution model to allow for a group of tasks to be scheduled. In section 4, we describe the extensions to the communication model that enable direct communications between servers. Section 5 gives a brief overview of mapping on a fully connected network, and the final section outlines the future work of this project.

2. Performance Models

The mapping of the individual tasks of the NetSolve application to computers of the network is based on the performance model of the heterogeneous network of computers and the performance model of the task. The heterogeneous network of computers is seen as a set of heterogeneous processors connected into a star with the client machine being the center of the star. The performance of each processor is characterized by the execution time of the same serial code multiplying two dense square matrices of some fixed size. This characteristic of the processor is updated periodically or whenever the load on the server changes beyond a certain threshold. The characteristics of the communication links are obtained using sensors that determine its latency and bandwidth. These characteristics are also updated periodically.

The performance model of an individual task is provided by the person who installs the task on the remote server. A formula is associated with each task that can be used to calculate the execution time of the task by the solver. The formula uses parameters of the task and the execution time of the standard computation unit on the computer (which is currently the multiplication of two dense square matrices of the fixed size). Apart from

the formula, the sizes of input and output data are also specified.

The mapping algorithm of NetSolve tries to minimize the total execution time of each individual task, which is calculated as the sum of the time of computation on the remote computer and the time of communications between the user's computer and the remote one to deliver input data and receive output data. To estimate the contribution of computation in the total execution time for each possible mapping, the agent evaluates the formula associated with corresponding remote task. To estimate the contribution of communication, the agent uses the current characteristics of the communication link between the user's computer and the remote computer, and the sizes of input and output data that should be transferred between the computers [2].

Conversely, the mapping algorithm of SmartNetSolve tries to minimize the overall execution time of a collective group of tasks. It does this by estimating the execution time of various mappings of a group of tasks. The estimated execution time of the mapping a group of tasks is based on the performance model of the group of tasks and the performance model of the fully connected heterogeneous network of computers.

2.1. Performance model of a group of tasks

In NetSolve, parameters, which characterize the performance model of an individual task, are specified by the person that installs the task, in a so called problem description file (PDF). The PDF syntax can only specify the performance model of an individual task and gives no means to specify the performance model of a group of tasks. A new programming interface is added to the client interface to allow the application programmer to specify the performance model of a group of tasks. This specification is written in a small dedicated specification language, Algorithm Definition Language (ADL).

ADL extends PDF, so that the application programmer can specify the collective performance model of a group of tasks by building on the performance model of individual tasks specified by software provider. In particular, ADL allows the application programmer to specify which tasks are in the group, the data dependencies between these tasks and the order of their execution. The ADL compiler will translate this specification into the code calculating the execution time of the group of tasks for each particular mapping of the tasks and virtual communication links between them. The code is used by the mapping component of SmartNetSolve, which is explained in more detail in section 3.

2.2. Performance model of a fully connected network

In NetSolve, the communication model only specifies the links of the star network, illustrated in Figure 1. These links are characterised by the latency and bandwidth between the server and the client, which are detected by NWS sensors. Such a sensor is started on the server after it is registered and the performance characteristics of the links are periodically updated. To employ the performance model of the fully connected network (illustrated in Figure 2), the performance model of the star network is extended and the characteristics of each link connecting all servers to each other are also determined. When a server is started the agent sends a list of the locations of all servers, the server then sends a sensor along these links and sends the results back to the agent. Consequently, the agent can assess the performance model of the fully connected network when mapping

3. Execution Model

Fundamentally, a NetSolve application can be thought of as number of ordered calls to execute tasks on remote servers. Two principal NetSolve calls are **netsl("taskname",...)** and **netslnb("taskname",...)**, which are blocking and non-blocking calls to perform the specified task **taskname**. Also included as parameters to these calls are pointers to the input and output data for the task. Consider a pseudo NetSolve application in Figure 3 consisting of three non-blocking tasks, *T1*, *T2* and *T3*, which are performed in parallel, and one blocking task *T4*. The first parameter of each of these calls refers to the task's name, the next parameter refers to the size of the input/output objects, the following are pointers to the inputs objects, and the last parameter points to the task's output object. In this case task *T4* has a data dependency on all three parallel tasks and therefore must wait until they have completed. Function **netslwt** is used to block the request of *T4* until these tasks have completed

```
if (x<10){
  request[0]=netslnb("T1", n, A, B);
}
  ...
  request[1]=netslnb("T2", n, C, D);
  request[2]=netslnb("T3", n, E, F);
  for(i=0;i<3<i++)
    info[i] = netslwt(request[i]);
  netsl("T4", n, B, D, F, G);
```

Figure 3. Sample NetSolve Application

```
smartMap(group1, x, n, … );
if (x<10){
    netslnb("T1", n, A, B);
}
  ...
netslnb("T2", n, C, D);
netsl("T3", n, E, F);

for(i=0;i<3<i++)
     info[i] = netslwt(request[i]);
netslnb("T4", n, B, D, F, G);
```

GROUP1

Figure 4. Sample SmartNetSolve Application

Each call to request the execution of a task in NetSolve essentially consists of two operations. The first is the mapping of that task to a server on the network and the second is the execution of the task on the server. When a request is issued, the client sends the task name to the agent. The agent then determines the servers that offer the fastest estimated execution time for the task based on the performance model of the task and network. A sorted list is then compiled of these "best" servers and this is sent to the client. The servers then receive input for the task, the task is executed, and the output data are returned to the client. The execution model of the application in Figure 3 is illustrated in Figure 5.

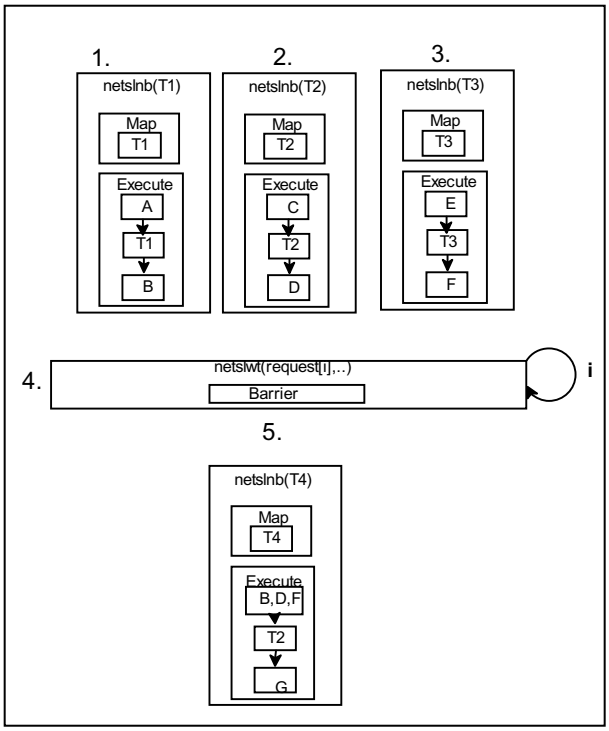

Figure 5. Execution model of NetSolve

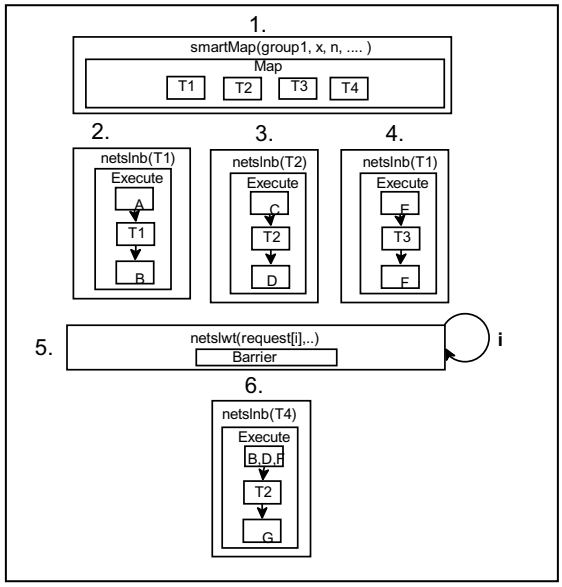

Figure 6. Execution model of SmartNetSolve

Consequently this execution model implies the mapping and execution of a task in NetSolve are completely inseparable, since they are initiated by the same request. Due to this constraint, the current execution model of the NetSolve system is unable to support the mapping of a group of tasks.

SmartNetSolve extends the NetSolve execution model so that the mapping of tasks can be done separately to the execution of the tasks, allowing a group of tasks to be collectively mapped before their execution is requested. In Figure 4, function **smartMap** maps the group of tasks **group1** to servers, and the following calls result in execution of the tasks of the group on those servers:

smartMap(nameOfGroup,listOfParameters...);

The first argument of **smartMap** specifies the name of the group of tasks to be mapped, and the others specify the parameters of the performance model of this group. When this function is called these parameters will be passed to the functions generated by the ADL compiler. This list of parameters resolves any ambiguity in the performance model of the group of tasks. For example, in the given application, it is not possible to determine if *T1* will be called until the value of **x** is determined. Therefore it is unclear whether *T4* has a direct dependency on *T1*. Furthermore, it is not possible to determine the sizes of the input and output objects of each task until the value of **n** is known. However, when the **smartMap** function is called the values of **x** and **n** are passed as parameters to the group's performance model, eliminating any ambiguity in the performance model, allowing the agent to estimate the execution time of a group of tasks for a given mapping. Figure 6

illustrates the execution of this application in SmartNetsolve. When the **smartMap** function is called all four tasks are mapped together. A mapping table is created, which will outline the details of the group's mapping. Based on this mapping, each task can be executed at each subsequent **netsl..()** call, without the need for any further mapping.

4. Communication Model

In SmartNetSolve, the communication model of NetSolve is extended to achieve the fully connected virtual network described in the introduction section of this paper. To establish this network, the SmartNetSolve system allows for direct communication between remote servers. A further extension to the communication model allows for broadcast communication to happen between remote servers and also between client and remote servers. The client-server communication model of NetSolve is realized by using a separate process called *proxy*. In NetSolve, the proxy process resides on the client computer and acts on behalf of the NetSolve client to handle all interactions with other NetSolve components [3]. With this implementation in NetSolve, the interactions are limited to only one proxy in any one session and also all interactions must involve the client. Figure 7 illustrates how the communication scheme would follow for the NetSolve application in Figure 3, given the value of **x** is less than 10. We assume *T1* is mapped to server *S1*, *T2* to server *S2*, *T3* to server *S3*, and *T4* to server *S4*.

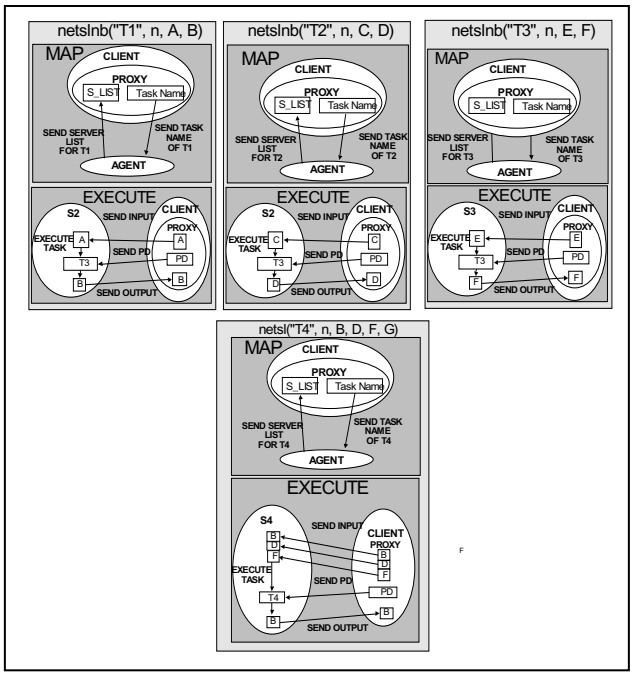

Figure 7. Communication of NetSolve

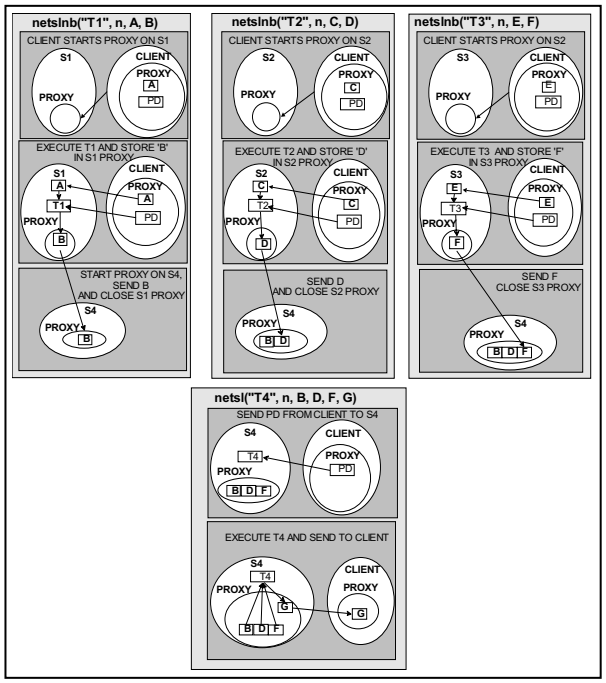

Figure 8. Communication of SmartNetSolve

Initially a proxy is started on the client machine and the client sends the name of the task to the agent. The agent then constructs a problem description object based on the problem description file of that task. Based on this problem description object, which represents the performance model of the task and the performance model of the network, the agent can determine a list of the most suitable servers for this task.

In addition to outlining the performance model of a task, the problem description object also outlines how the task should be executed, detailing such task characteristics as the path name of the task and the path of any libraries used. Therefore when this problem description object is sent to the server, the server has all the attributes necessary to execute the task. When the server finishes executing the task, it sends the result back to the client.

In the SmartNetSolve system, the server is extended to provide the following new capabilities that extend the core services of NetSolve server:

- The client may start a proxy on a remote server,
- The remote proxy may store data until it is needed,
- The remote proxy may send data to another remote proxy
- The server may start a proxy on another remote server.

The problem description object is extended to embody attributes, which aid the server-server

communication of NetSolve Objects. In addition the problem description of each task will now store the location of where the outputs of the task should be sent.

Figure 8 illustrates the functionality of this extended communication model for the sample SmartNetSolve application in Figure 4, given the value of **x** is less than 10. Again, we assume the agent maps *T1* to *S1*, *T2* to *S2*, *T3* to *S3*, *T4* to *S4* and maps each virtual link between tasks *T1*, *T2*, *T3* and task *T4* on to the path that directly connects their respective servers. This diagram does not feature any mapping since this is done prior to the request of any task. The mapping table, which is used by the client when executing tasks, is produced when the **smartMap** function is called.

In Figure 8 it is evident that when each of the non-blocking tasks is requested, the client will initially set up a proxy on each of the servers which is executing the parallel tasks. This is because the client is aware from the mapping table that the result of each of these tasks must be sent to another server and not back to the client. This proxy is set up for the purpose of storing the result of these tasks and to give the server the capability of handling the interaction with the other dependent server. Once this proxy is set up the client sends the input and problem description object to each of the servers. Upon analysis of the problem description object, each server will determine that its result is required by S4. When each of the tasks is finished their corresponding server will determine if there is a proxy started on S4. If not it will start one before sending its output. Once the outputs have been sent, each server will close their own local proxy.

When each of the three parallel tasks is completed, *T4* is requested. The client does not send the server any input as it is already residing on the server's proxy. It merely sends the problem description object for the task. Upon analysis of the problem description object, the remote server will determine that its result must be sent to the client. S4 can now complete its task and return the result to the client.

5. Mapping

The objective of the NetSolve mapping algorithm is to find the mapping for each individual task that minimizes the execution time for that task. This mapping involves the assignment of the individual task to a server on the network. The goal of the mapping algorithm of SmartNetSolve is to find the mapping for a collective group of tasks that minimizes the execution time for that group. Mapping in SmartNetSolve not only involves the mapping of each task in the group to a server on the network but also the mapping of virtual links between tasks to communication paths of the network. This mapping allows further optimization by minimizing the total communication time of the group. For each given task-to-server mapping, the mapping algorithm can minimize the overall communication time by mapping the virtual links between dependent tasks to the shortest communication path between the remote tasks on network. The worst case complexity of finding these shortest paths is $O(n^2)$ [7]. The mapping algorithm finds the optimal mapping by comparing the summation of the total estimated computation and communication time of each given group mapping.

Calculating the estimated time for every possible solution when mapping a group of tasks becomes very time consuming when the number of tasks and servers increase beyond a few. A number of heuristics finding suboptimal mappings will be implemented and tested.

5.1. Comparison of NetSolve and SmartNetSolve mappings

For the purpose of demonstration, we compare a hypothetical mapping of the NetSolve application in Figure 3 and a mapping for the corresponding SmartNetSolve application in Figure 4, given that the value of **x** is less than 10. We will make the following assumptions

- The relative size of tasks *T1*, *T2* and *T3* are as follows: *T1<T2<<T3*
- There is only one server, *S4*, that is capable of executing *T4* and there are three servers, *S1*, *S2*, *S3*, that are available in the network capable of executing tasks *T1*, *T2*, *T3*. The relative speeds of these servers are as follows: *S1<S2<<S3*.

Figure 9a shows how NetSolve would map this application. When NetSolve maps an application, it only optimises the execution time of each individual task, based on the performance model of that task and the performance model of the network at that point in time. In other words, the system is unaware of what tasks are to follow when each task is mapped.

As a result, the first and smallest task *T1* would be mapped to the fastest server *S3* as this would yield the lowest execution time. At this point, the system is unaware that two larger tasks will be executed in parallel with this task. Following the mapping of *T1*, the system maps *T2* and it would be assigned the second fastest server *S2*, as *T1* is currently executing on *S3*. Consequently *T3*, the largest task would then be mapped to the slowest server, *S1*, since *S3* and *S2* are now both busy with the other two tasks. This mapping has a great consequence to the overall computation time of the group of tasks. This is highlighted in Figure 10a.

Since NetSolve is restricted to the client-server model, this mapping would result in a communication pattern

that would involve the client in all data transfers. When parallel tasks *T1*, *T2*, and *T3* are finished, their results must be sent back to the client. These results are needed by *T4*, so the client then sends them to *S4* when *T4* is requested. This unnecessary bridge communication with the client increases the overall communication time of the group. The bridge communication is highlighted in Figure 9a and its overall contribution to the execution time of the group is highlighted in Figure 10a.

Figure 9b shows how SmartNetSolve would map this application, given the performance model of the collective group of tasks and the fully connected network. With these performance models, the SmartNetSolve system would be fully aware of each task in the group, the virtual links between these tasks and can therefore choose a more optimal mapping for the group. Given this knowledge, the mapping algorithm will map the tasks to servers and virtual links to communication paths on the network in such a way as to optimise the total computation and communication of the group.

The shaded boxes in Figure 9b show the mapping algorithm optimising the total computation time for the group of parallel tasks *T1*, *T2*, and *T3*. This task-to-server mapping implements an improved load balancing strategy, mapping the largest task to the fastest server and vice versa. This reduces the overall computation time for this group of parallel tasks. This improvement is highlighted in Figure 10b.

Further optimisation is achieved by mapping each of the virtual links *T1→T4*, *T2→T4*, and *T3→T4* to the fastest communication path connecting the corresponding servers. In this case, the mapping algorithm maps the virtual link to the communication path directly connecting the servers, which eliminates any bridge communication with the client. This improvement in communication time is also highlighted in Figure 10b.

In this hypothetical example, the improved and more intelligent mapping algorithm of SmartNetSolve has given a 3 fold speed up.

6. Conclusion and future work

In this paper, we have presented an approach to extending the RPC model of NetSolve to allow the mapping of a group of tasks on a fully connected virtual network. At present, the execution model has been extended successfully to achieve the communication between remote servers. The execution model has also been extended to allow for mapping of a group of tasks. We are currently working on employing the two extended performance models. The ADL language, which is used to specify the performance model of a group of tasks, and the NWS sensors, which will be used to employ the performance model of the fully connected network, are currently under implementation. In the future, a number of mapping heuristics will be implemented and tested to determine which would produce better suboptimal mappings.

Acknowledgements

The authors would like to thank Jack Dongarra for valuable discussions and support in this project. The work was supported by the Science Foundation Ireland and also in part by the IBM Dublin Center for Advanced Studies.

References

[1] K. Seymour, A. YarKhan, S. Agrawal, and J. Dongarra. NetSolve: Grid Enabling Scientific Computing Environments. Computer Science Department, University of Tennessee, Knoxville, TN 37919, USA, 2005.

[2] J. Dongarra, A. Lastovetsky. An Overview of Heterogeneous High Performance and Grid Computing. In *Engineering the Grid: Status and Perspectives*, Eds. B. Di Martino, J. Dongarra, A. Hoisie, L. Yang, and H. Zima, American Scientific Publishers, January 2006.

[3] D. Arnold, H. Casanova, J. Dongarra. Innovations of the NetSolve Grid Computing System. *Concurrency and Computation: Practice and Experience*, 14(13-15):1457-1479, Wiley, 2002.

[4] A. Lastovetsky. *Parallel Computing on Heterogeneous Networks*, 423 pp., Wiley, June 2003.

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

[6] K. Taura, A. Chien. A Heuristic Algorithm for Mapping Communicating Tasks on Heterogeneous Resource. *Proceedings of the 9th Heterogeneous Computing Workshop* (HCW 2000), pp.102-115, IEEE Computer Society Press, 2000.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

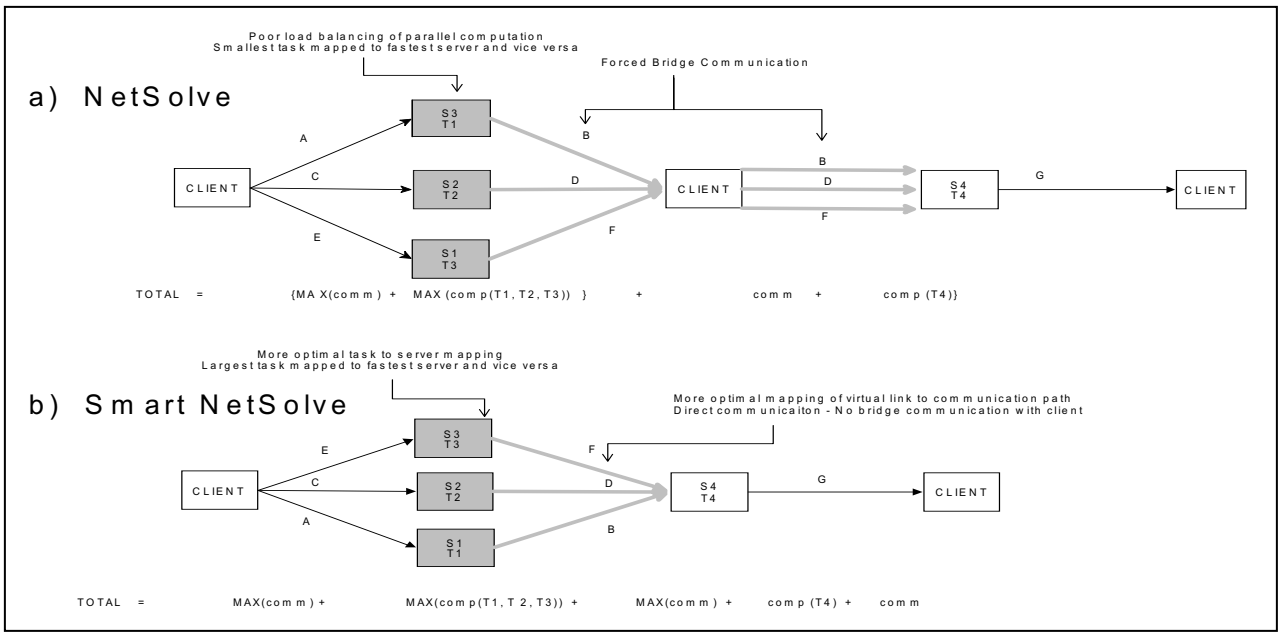

Figure 9. a) Mapping of an application in NetSolve b) Mapping of an application in SmartNetSolve

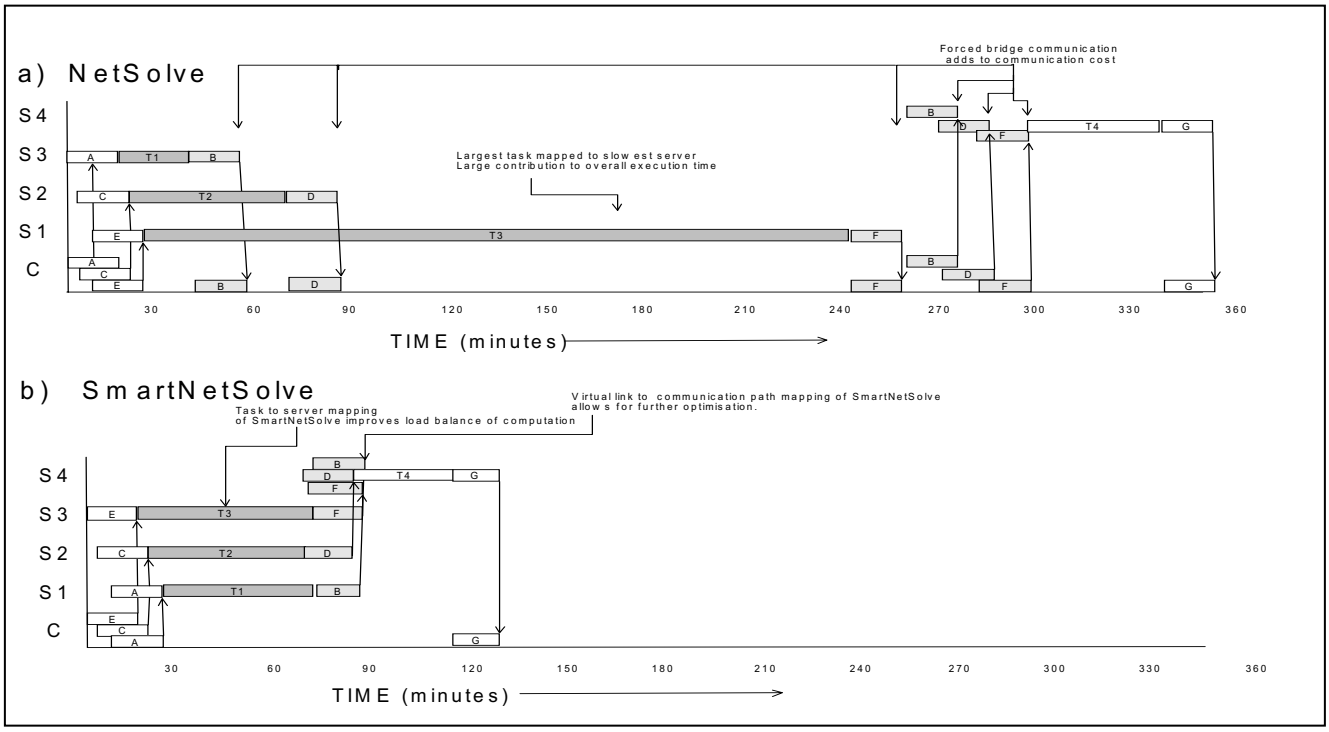

Figure 10 a) Timing of the mapping of NetSolve b) Timing of the mapping of SmartNetSolve