Implementing Effective Data Management Policies in Distributed and Grid Computing Environments[⋆]

Luisa Carracciuolo[1], Giuliano Laccetti[2], and Marco Lapegna[2]

[1] Institute of Chemistry and Technology of Polymers (ICTP)
National Research Council (CNR)
c/o Department of Chemistry
via Cintia Monte S. Angelo, 80126 Naples, Italy
`lcarracci@unina.it`

[2] Department of Mathematics and Applications
University of Naples Federico II
via Cintia Monte S. Angelo, 80126 Naples, Italy
`{giuliano.laccetti,marco.lapegna}@dma.unina.it`

Abstract. A common programming model in distributed and grid computing is the client/server paradigm, where the client submits requests to several geographically remote servers for executing already deployed applications on its own data. In this context it is mandatory to avoid unnecessary data transfer because the data set can be very large. This work addresses the problem of implementing a strategy for data management in case of data dependencies among subproblems in the same application. To this purpose, some minor changes have been introduced to the Distributed Storage Infrastructure in NetSolve distributed computing environment.

1 Introduction

As stated in [8], a computational grid is a system that coordinates resources that are not subject to centralized control, using standard, open, general purpose protocols and interfaces, to deliver non trivial qualities of services. That means that a Grid infrastructure is built on the top of a collection of disparate and distributed resources (computers, databases, network, software, storage) with functionalities greater than the simple sum of those addends [9]. The added value is a software architecture aimed to virtualize scattered computing and data resources to create a single computing system image, granting users and applications seamless access to vast IT capabilities. The hardware of this single computing system is often characterized by slow and non dedicated Wide Area

⋆ This work has been carried out as part of the project "Sistema Cooperativo Distribuito ad Alte Prestazioni per elaborazioni Scientifiche Multidisciplinari (S.CO.P.E.)", supported by the Italian Ministry of Education, University and Research (MIUR), PON 2000-2006.

Networks (WAN) connecting very fast and powerful processing nodes (that can be also represented by supercomputers or large clusters) scattered on a huge geographical territory, whereas the operating system (the grid middleware) is responsible to find and allocate resources to the scientists applications, taking into account the status of the whole grid. Many papers focus on this aspect of the grid computing, addressing topics, such as resources brokering (e.g. [4], [14]), performance contract definition and monitoring (e.g. [10], [12]) and migration of the applications in case of contract violations (e.g. [15]).

On the other hand, common scientific applications are characterized by very large input data and dependencies among subproblems, thus it is not sufficient to choose the most powerful computational resources in order to achieve good performance, but it is essential to define suitable methodologies to distribute application data onto the grid components overlapping communication and computation and to provide tools that eliminate unnecessary data transfers. In this research area a small number of papers are available (e.g. [7]).

However, because of the natural vision of a computational grid as a single computational resource, it is possible to borrow ideas and methodologies commonly utilized for traditional systems and to adapt them to new environments. This paper is structured as follows: in section 2 we describe our caching methodologies to address data distribution onto the grid components in case of significant dependencies among subproblems of a scientific application; in section 3 we describe the software environment we use with some minor modifications we introduced to better adapt it to our aims; in section 4 we show the experiments we carried out to validate our methodology ; finally in section 5 we provide conclusions and outline our future works.

2 Data Management in Distributed Environments

For many years the dominant style in parallel computing has been the Single Program Multiple Data programming model, where all processors use the same program, although each has its own data. The algorithms based on these techniques are designed to be used in a static and dedicated computing environment, like an MPP supercomputer. They need also frequent data communications and synchronizations among homogeneous nodes in a systolic fashion, where it is also possible to overlap communication and computation. A notable example is the ScaLAPACK library [3] where the data are mapped on a 2D grid of processors. A computational grid is a computing environment which differs substantially from the one previously described. In this case the heterogeneity and the sharing capability of the resources make the client/server programming model more promising compared to the SPMD approach, as it eliminates the need of data communication among the server nodes. Probably the most famous project based on this model is the SETI@home project designed to find extraterrestrial intelligence, where a central system send a huge number of independent tasks to the participating servers for the computation [13]. In the client/server programming model the data are stored in the client from where are sent in chunks to the servers for computations; after the computation the results are returned to the client. The data movement from client and servers in a grid is similar to the data transfer between memories and processing unit in a single Non Uniform Memory Access (NUMA) machine. A memory-aware model of the grid is shown in Fig. 1, where the computational units (the servers) can retrieve data from registers and caches, main memory, secondary storage, as well as from remote clients. Fast and small memories are positioned at the higher level, whereas slower memories, that are usually accessed by means of geographic networks, are located at the lower ones. In this model the servers' secondary storage level can be represented by the disks of each server as well as by an external (to the server) data repository, however close enough to make the access to this level negligible for the remote client.

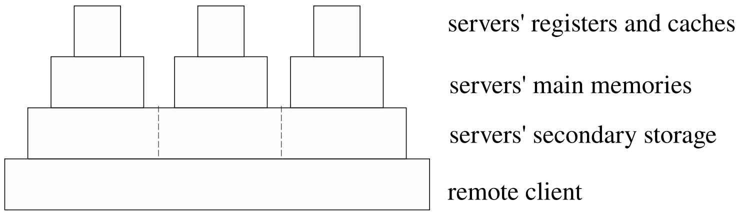

Fig. 1. A memory-aware model of a computational grid

Fig. 2 shows typical peak bandwidth and latency of the three lowest memory levels when accessed from the server. The illustrated values refer to a common workstation usually available in a distributed computing environment and are not representative of the leading edge technology. For the server main memory the values for a DDR2 memory interface running at 400 MHz are reported; for the server secondary storage the values for a Parallel ATA disk adapter are reported; for the remote client both values for a Local Area Network (a Fast Ethernet) and for a Wide Area Network (e.g. a metropolitan area network) are reported. Note however that the LAN bandwidth is shared among all the data transfers on the network, so that the actual bandwidth is very sensitive to the network traffic and can be significantly below the peak bandwidth. This performance reduction is much less evident for the disk transfer rate because of the optimization of the disk scheduling algorithms in the current operating systems.

It is commonly acknowledged that the key strategy to achieve high performances with a NUMA machine is an extensive use of caching methodologies at

	Bandwidth	Latency
Server main memory	10 GByte/sec	2-10 ns
Server secondary storage	100 MByte/sec	5 ms
Remote client (LAN)	12.5 MByte/sec	10 ms
Remote client (WAN)	$<$ 1 MByte/sec	100 msec

Fig. 2. Typical values for bandwidth and latency in Fig. 1

each level of the memory hierarchy, in order to provide the computing elements with data taken from fast memories at the high level and to avoid unnecessary data transfer toward the lowest levels. Compilers or problem-oriented libraries, like the BLAS library for numerical linear algebra [6], are usually in charge of the management of the higher levels of the memory hierarchy, but the lowest levels have to be managed by means of suitable programming methodologies and software tools. Scientific applications rarely can be divided in totally independent tasks and some data dependencies are always present among them, thus, the definition of methodologies and the development of software tools for an effective data distribution among the components of a grid assume a key role in grid computing. As an example, assume an application composed by three tasks with dependencies in the form of three pipelined stages, as shown in Fig. 3. In this example the output data from stage 1 represent the input data for stage 2, and the output data from stage 2 represent, in turn, the input data for stage 3.

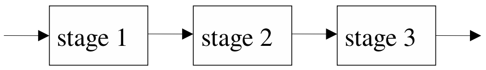

Fig. 3. A pipelined three stages application

A raw implementation of this application with the client/server programming model is depicted in Fig. 4 where three servers compute the three stages of the application. In this implementation the output data from stages 1 and 2 are sent back to the client and then sent again to a new server for the computation of the next stage. In this case the input data for stages 2 and 3 will be located at the lowest level of the memory hierarchy in Fig. 1 when accessed by the servers.

In Fig. 5, the use of the server secondary storage as a cache for the intermediate results allows to locate them to a higher level in the memory hierarchy and avoids unnecessary data transfers toward the client memory. Furthermore, by keeping intermediate data in higher level memories it's possible to overlap data

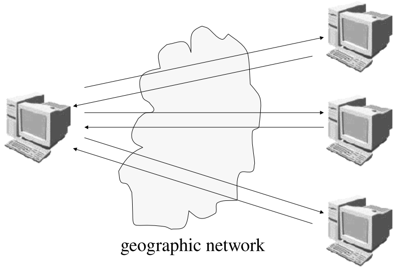

Fig. 4. Raw implementation of the application in Fig. 2

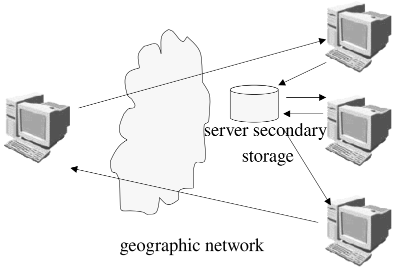

Fig. 5. Implementation of the application in Fig. 2 with caching of the intermediate results

communication and stage computation if the entire sequence has to be repeated several times. A similar approach to the data management in distributed environments is described in [7], where the server main memory is used as a cache in place of the server secondary storage. The main advantage of the approach described in the current section is the larger amount of available space to cache the intermediate data, with an access time to the cache however negligible respect to the client memory.

3 Software Tools for Caching Data in NetSolve

One of the best known software environments for the on demand approach to distributed and grid computing is NetSolve 2.0 [1]. This is a software environment based on a client-agent-server paradigm, that provides a transparent and inexpensive access to remote hardware and software resources. In this environment a key role is played by the agent, that collects hardware performance and available software of the servers in the environment setup phase as well as dynamic information about the workload of the resources. When the agent is contacted by the client by means of the NetSolve client library linked to the user application, it selects the most suitable server to be used on the basis of stored information and notifies the client. Therefore, the client can send data directly to the selected server that performs the computation by using a code generated through a Problem Description File that acts as interface between NetSolve and the legacy software. Finally the result is directly sent back to the client. This data exchange protocol is executed for every request to NetSolve, then in case of dependency among multiple tasks in the same application, the execution looks like those in Fig. 4, with a superfluous network traffic. NetSolve however includes two tools in order to manage data more efficiently: the Request Sequencing and the Data Storage Infrastructure.

– The Request Sequencing is realized by means of an appropriate NetSolve construct that builds a Direct Acyclic Graph where the nodes represent the tasks and the arcs represent the data dependencies among them. The main limitation of this approach is that currently the entire DAG sequence is executed by the same server, even if there are independent tasks in the sequence that can be executed in parallel by multiple servers.
– The Distributed Storage Infrastructure (DSI) is an attempt to overcome the limitation of the Request Sequencing in NetSolve, by allowing the client to place data in a storage infrastructure that can be accessed by the server. If the storage resource is sufficiently close to the servers, it's possible to reduce the network traffic in case of data dependency among tasks. Currently NetSolve implements this storage service by means of the Internet Backplane Protocol (IBP) [2],[11], a middleware for managing and using remote resources. Fig. 6 shows how the network traffic is reduced in case of multiple accesses to the stored data by sending them only once from client to the computational space formed by the servers and IBP storage. However, this approach also shows currently a drawback: the servers are able to read data

from the IBP storage but they appear to be unable to write on it, so that the implementation in Fig. 5 can be only partially achieved.

In order to fully implement a caching methodology like those shown in Fig. 5, it has been necessary to modify in some extent the NetSolve DSI implementation. More precisely, the DSI infrastructure defines the new data type DSI_OBJECT as a data structure describing the location and several information about the IBP storage area (dimension, access permissions, access mode,) that can be used as a cache. In a typical NetSolve session, a DSI_OBJECT is generated by the client and sent to the servers by means of the NetSolve API, so that they can access the IBP storage. Among the information in a DSI_OBJECT there are the read/write/management capabilities of the IBP storage, i.e. unique character strings used as keys to access correctly the data on the storage.

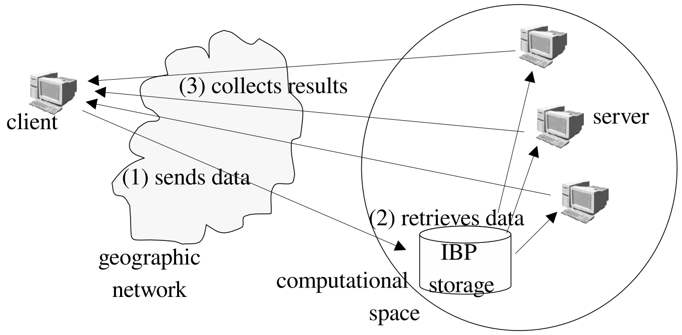

Fig. 6. Implementation of the Distributed Storage Infrastructure in NetSolve

The main changes are therefore related to the DSI functions for reading and writing data on the IBP storage, so that they can be used also by the servers. Actually, at the present time the servers cannot call directly these DSI functions because the Problem Description Files used to generate the server codes are unable to manage a DSI_OBJECT. For this reason in the modified implementation of the DSI infrastructure, the APIs of the DSI functions for accessing the IBP storage include the capabilities of the IBP storage in place of the DSI_OBJECT. The character type used for the IBP capabilities is managed by the Problem Description File, thus the servers can use without restriction the DSI function for reading and writing on the data storage. As an example, consider the API of the current DSI function to read a vector from the IBP storage:

int ns_dsi_read_vector(DSI_OBJECT* dsi_obj, void* data, int count, int data_type)

It has been modified in:

int ns_new_read_vector(char* ibp_read_cap, void* data, int count, int data_type)

where ibp_read_cap is the IBP capability to read data in the storage area. Similar changes have been carried out to the functions ns_dsi_write_vector(), ns_dsi_read_matrix(), ns_dsi_read_matrix() and ns_dsi_close(). As a consequence, it has been necessary to introduce some minor changes in the related code, and the software infrastructure obtained with the modified functions has been used in place of the DSI in the NetSolve architecture.

4 Computational Experiments

As a test bed for the software infrastructure described in the previous section a block matrix multiplication algorithm has been chosen because it is a basic linear algebra computational kernel which is representative of similar other computations; on the other hand, it encompasses a lot of data movements, and in such a case, minimizing communication overhead becomes a challenging task. More precisely, the client/server *ijk form* of the block matrix algorithm has been used, because it exploits a smaller synchronization overhead compared to the other forms [5]. For simplicity, assume that A, B and C are square matrices of order n, and divided in square blocks $C(I, J)$, $A(I, K)$ and $B(K, J)$ of order r, with n divisible by r, so that let NB the number of blocks in each dimension, it is $NB = n/r$.

```
for I=1, NB (in parallel)
  for J=1, NB (in parallel)
   choose a server
   for K=1, NB
    send C(I,J), A(I,K), B(K,J)
                 to the server
    receive C(I,J) from the server
   endfor
  endfor
endfor
```

client algorithm

```
receive C(I,J), A(I,K), B(K,J)
              from the client
C(I,J) = C(I,J) + A(I,K), B(K,J)
send C(I,J) to the client
```

server algorithm

Fig. 7. The client server *ijk form* of the block matrix multiplication algorithm (Algorithm 1)

In Algorithm 1 (Fig. 7) the client can manage NB^2 independent tasks over the indices I and J and synchronizations occur only among two successive values of K in the same task. Furthermore, the computation of a single matrix product in the server can be performed through the BLAS3 sequential DGEMM routine [6]. Let now T_s and T_r be respectively the access time to the server secondary storage and to the remote client memories. The communication cost for the complete computation of each block $C(I, J)$ with Algorithm 1 is then:

$$T_1 = 4NB\, T_r\, r^2.$$

However in the Algorithm 1 it is obvious that in the innermost loop of the client algorithm, the same block $C(I, J)$ is received and sent again to the server for successive values of the index K. This is an example of unnecessary data movement between client and server that can be avoided by storing intermediate results in the server secondary storage. A more efficient solution is therefore the Algorithm 2 (Fig. 8). From the bandwidth values in Fig. 2 is $T_s < T_r$, so the communication cost for the computation of each block $C(I, J)$ with Algorithm 2 is then :

$$T_2 = 2NB\, r^2\ (T_s + T_r) < T_1.$$

To test the software infrastructure needed to implement the data management policy described in Section 2, some experiments have been carried out on a cluster of 2.4 GHz PCs, each of them provided with a Parallel ATA disk adapter with a peak transfer rate of 100 MByte/sec, and connected using a 100 Mbits switch. The operating system running on the PCs was Linux 2.4. Algorithm 1 and Algorithm 2 have been implemented by using NetSolve-2.0 computing environment with the DSI infrastructure modified as described in Section 3. The IBP 1.0.4.2 software infrastructure has been installed on the servers of the NetSolve system in order to support the modified DSI infrastructure.

```
for I=1, NB (in parallel)
  for J=1, NB (in parallel)
  choose a server
  store C(I,J) in the server
          secondary storage
   for K=1, NB
    send A(I,K), B(K,J)
          to the server
    endfor
   retrieve C(I,J) from the server
  endfor
endfor

        client algorithm
```

```
retrieve C(I,J) from the
        secondary storage
receive A(I,K), B(K,J) from client
C(I,J) = C(I,J) + A(I,K), B(K,J)
store C(I,J) in the
        secondary storage

        server algorithm
```

Fig. 8. The client server ijk form of the block matrix multiplication algorithm with caching of intermediate results in the server secondary storage (Algorithm 2)

Fig. 9 shows the total execution times in seconds for Algorithm 1 and Algorithm 2 for matrices of order $n = 50, 100, 200$ and 300 with square blocks of order $r = 50$.

In order to minimize the impact of the traffic fluctuation in the network, the reported values are the average times over 10 runs. The results show a significant reduction of the total execution time for the computation of the entire matrix multiplication. Same results are obtained also with larger problems: Fig. 10 shows the total execution times of Algorithm 1 and Algorithm 2 in case of matrices of order $n = 500, 1000$ and 2000 with square blocks of order $r = 500$.

	Algorithm 1	Algorithm 2
n=50 (NB=1)	0.08	0.08
n=100 (NB=2)	0.41	0.36
n=200 (NB=4)	9.5	5.2
n=300 (NB=6)	14.38	8.5

Fig. 9. Execution times for a block matrix multiplication with square blocks of order $r = 50$

	Algorithm 1	Algorithm 2
n=500 (NB=1)	2.46	1.81
n=1000 (NB=2)	17.7	12.2
n=2000 (NB=4)	66.6	44.4

Fig. 10. Execution times for a block matrix multiplication with square blocks of order $r = 500$

5 Conclusions and Future Works

In this work, it is addressed the problem of implementing a strategy for data management in case of data dependencies among subproblems in the same application. The main aim of this paper is twice. On one side, an effective methodology for the placement of data among the resources of a distributed environment with the client/server programming model has been described. The methodology is based on the observation that a computational grid is a large NUMA machine, where at the lowest memory level there is the client and at the highest level there are the server resources. Therefore an effective client/server implementation needs that the application data have to be as more as possible close to the servers. On the other hand it has been necessary to modify in some extents the Distributed Software Infrastructure, part of the NetSolve distributed computing system, in order to implement the described methodology. The computational experiments confirm the expectations, showing a significant reduction of the execution times when the intermediate data are kept in the secondary storage of the servers. Future works are devoted to integrate the obtained software infrastructure in the grid environment of an ongoing project conducted by the University of Naples Federico II and funded by the Italian Ministry of University and Research. The main aim of the project is to solve multidisciplinary applications, deriving from Naples scientists researches, in a new powerful grid infrastructure to be integrated in large national and European grids.

References

1. Arnold, D., Agrawal, S., Blackford, S., Dongarra, J., Miller, M., Seymour, K., Sagi, K., Shi, Z., Vadhiyar, S.: User's Guide to NetSolve V. 2.0. Univ. of Tennessee, See also NetSolve home page (2004), `http://icl.cs.utk.edu/netsolve/index.html`

2. Bassi, A., Beck, M., Moore, T., Plank, J.S., Swany, M., Wolski, R., Fagg, G.: The Internet Backplane Protocol:a study in Resource Sharing. Future Gener. Comput. Syst. 19, 551–561 (2003)
3. Blackford, L., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D.,, R.: Whaley: ScaLAPACK Users Guide. SIAM, Philadelphia (1997)
4. Czajkowsky, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S.: A Resource Selection Management Architecture for Metacomputing Systems. In: Feitelson, D.G., Rudolph, L. (eds.) IPPS-WS 1998, SPDP-WS 1998, and JSSPP 1998. LNCS, vol. 1459, pp. 62–82. Springer, Heidelberg (1998)
5. D'Amore, L., Laccetti, G., Lapegna, M.: Block matrix multiplication in a distributed computing environment: experiments with NetSolve. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 625–632. Springer, Heidelberg (2006)
6. Dongarra, J.J., Du Croz, J., Hammarling, S., Hanson, R.J.: A Proposal for an Extended Set of Fortran Basic Linear Algebra Subprograms. ACM SIGNUM Newsletter 20, 2–18 (1985)
7. Dongarra, J.J., Pineau, J.F., Robert, Y., Shi, Z., Vivien, F.: Revisiting Matrix Product on Master-Worker Platforms. In: APDCM workshop IPDPS 2007 Conference. A revised version is in International J. on Foundations of Computer Science (in press, 2007)
8. Foster, I.: What is the Grid? A three point checklist, `http://www-fp.mcs.anl.gov/~foster/Article/WhatIsTheGrid.pdf`
9. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan and Kaufman, San Francisco (1998)
10. Petitet, F., Blackford, S., Dongarra, J., Ellis, B., Fagg, G., Roche, K., Vadhiyar, S.: Numerical Libraries and the Grid:The GrADS Experiment with ScaLAPACK. University of Tennessee Technical Report UT-CS-01-460 (2001)
11. Planck, J., Bassi, A., Beck, M., Moore, T., Swany, M., Wolsky, R.: Managing Data Storage in the Network. IEEE Internet Computing 5, 50–58 (2001)
12. Ribler, R., Vetter, J., Simitci, H.,, D.: Reed: Autopilot:Adaptive Control of Distributed Applications. In: The 7th IEEE High Performance Distributed Computing Conference, pp. 172–179. IEEE Computer Society, Los Alamitos (1998)
13. Seti@home home page: `http://setiathome.ssl.berkeley.edu/`
14. Vadhiar, S., Dongarra, J.: A Metascheduler for the Grid. In: The 11th IEEE High Performance Distributed Computing Conference, pp. 343–351. IEEE Computer Society, Los Alamitos (2002)
15. Vadhiar, S., Dongarra, J.: A performance oriented migration framework for the grid. In: CCGRID 2003 The 3rd International Symposium on Cluster Computing and the Grid, pp. 130–137. IEEE Computer Society, Los Alamitos (2003)