Loosely-coupled Loop Scheduling in Computational Grids *

J. Herrera[1], E. Huedo[2], R.S. Montero[1], I.M. Llorente[1,2]

[1] Univ. Complutense de Madrid
Dep. Arquitec. de Comp. y Automática
Facultad de Informática 28040, Spain

[2]Cent. Astrobiología (CSIC-INTA)
Lab. Computación Avanzada
Simulación y Aplicaciones Telemáticas
28850 Torrejón de Ardoz, Spain

Abstract

Loop distribution is one of the most useful techniques to reduce the execution time of parallel applications. Traditionally, loop scheduling algorithms are implemented based on parallel programming paradigms such as MPI. This approximation presents three main disadvantages when applied in a Grid environment, namely: (i) all resources must be simultaneously allocated to begin execution of the application; (ii) it is necessary to restart the whole application when a resource fails; (iii) it is not possible to add new resources to a currently running application. To overcome these limitations, we propose a new approach to implement loop distribution schemes in computational Grids. This approach is implemented using the Distributed Resource Management Application API (DRMAA) standard and the GridWay meta-scheduling framework. The efficiency of this approach to solve the Mandelbrot set problem is analyzed in a Globus-based research testbed.

1 Introduction

Since the end of the eighties, with the introduction of distributed memory massive parallel systems (MPPs), a great effort has been invested in exploiting the potential computing power of these parallel computers. New programming paradigms and performance theories were developed, and systematically adapted or extended as new architectures appeared, like symmetric multiprocessors (SMP) or clusters. Traditionally, for all these parallel architectures, the distribution of program loops among computing resources has been proved as an efficient approach to extract the parallelism of applications. Loops constitute an important source of parallelism in a program, and when there is no dependencies between iterations (*parallel loops*) a high efficiency can be obtained.

In this way, a great research effort has been made on parallel loop scheduling (*self-scheduling*). Several self-scheduling strategies has been devised and successfully used in "traditional" parallel systems (MPPs and SMPs) namely: static scheduling schemes for loops with iterations with an homogenous workload; and dynamic scheduling schemes to prevent load unbalance in heterogenous workload loops [1]. These *simple* self-scheduling schemes (following the notation used in [1]) has been later extended to deal with heterogeneous cluster architectures, considering processor speeds and load [2].

Grid computing has emerged in the last decade as a promising computing platform that can offer a dramatic increase in the number of available processing resources that can be delivered to applications. In relation to the previous parallel systems, the Grid presents some genuine characteristics, namely: a high degree of heterogeneity, high fault rate and dynamic resource availability. Therefore, an efficient Grid self-scheduling scheme should consider these characteristics. In this context, K. Cheng et al. [3] have proposed a two-phase self-scheduling scheme that only takes into account the heterogeneity of Grid CPUs.

In this work we propose a loosely-coupled self-scheduling strategy for Computational Grids. This new strategy presents some benefits compared with the traditional approach followed in the literature and establish the basis to deal with all the Grid characteristics mentioned above. We also assess the impact of Grid characteristics, which has not been previously considered in self-scheduling. To this end, we consider the simple self-scheduling schemes to distribute

*This research was supported by Ministerio de Ciencia y Tecnología, through the research grant TIC 2003-01321 and 2002-12422-E, and by Instituto Nacional de Técnica Aeroespacial "Esteban Terradas" (INTA) – Centro de Astrobiología.

the Mandelbrot set application on a slightly heterogeneous testbed based on the Globus Toolkit.

The organization of the paper is as follows. In Section 2 we briefly review the self-scheduling schemes used in this paper. Section 3 present the computational Grid environment of our research. Then, Section 4 describes a loosely-coupled implementation of simple self-scheduling schemes in a Grid environment. In Section 5 we present experimental results of the execution of the Mandelbrot set application on a research testbed. Finally, we end this paper with some conclusions.

2 Dynamic Self-Scheduling Schemes

In this work, we will focus on dynamic schedulers because these algorithms are more suitable to heterogeneous environments [1]. In particular, we will study the behavior of simple self-schedulers, also refereed as self-scheduling loops [2]. In general, self-scheduling techniques are based on the master-worker paradigm. In a master-worker model, there is a node (master) which dynamically assigns tasks to the rest of the worker nodes. When a worker node ends its execution, it sends the results to the master and requests new tasks.

In the following list we describe the most frequently used self-scheduling schemes. Let I the total number of iterations, C_i the iterations assigned to each processor, and R_i the remaining number of iterations:

- *Chunk Self-Scheduling (CSS).* The chunk size is fixed:

$$C_i = chunk \text{ with } chunk \geq 1 \text{ and } R_i = R_{i-1} - C_i. \quad (1)$$

- *Guided Self-Scheduling (GSS).* The user can choose the minimum chunk-size assigned to each processor. The chunk size is determined by dividing the number of remaining iterations by the number of available processors.

$$C_i = \left\lceil \frac{R_{i-1}}{p} \right\rceil \text{ with } R_i = R_{i-1} - C_i. \quad (2)$$

- *Trapezoid Self-Scheduling (TSS).* The chunk size is linearly decreased a given amount (D), with the first and last chunk sizes fixed (F, L), N represents the number of task assigned:

$$C_i = C_{i-1} - D \text{ with } D = \left\lfloor \frac{(F-L)}{(N-1)} \right\rfloor \text{ and } N = \left\lceil \frac{2I}{F+L} \right\rceil. \quad (3)$$

The F and L parameters can be defined by the user or:

$$F = \frac{I}{2p} \text{ and } L = 1. \quad (4)$$

- *Fixed Increase Self-Scheduling (FISS).* Where X is a user parameter and σ is the number of stages. B represents chunk increase or 'bump':

$$C_i = C_{i-1} + B \text{ with } C_0 = \left\lfloor \frac{I}{Xp} \right\rfloor \text{ and } B = \left\lceil \frac{2I(1-\sigma/X)}{p\sigma(\sigma-1)} \right\rceil \text{ and } X = \sigma + 2 \quad (5)$$

Table 1 shows the different chunk sizes (C_i) generated by the previous self-scheduling algorithms for a problem with 5 nodes ($p = 5$), and $I = 50000$.

3 Computational Grid Environment

A Grid is a computational environment that coordinates resources that are not subject to centralized control, using standard, open, general-purpose protocols and provides nontrivial qualities of service. In general, Grid technology is based on four layers, namely: (i) *Grid fabric*, Grid resources interconnected by a heterogeneous network; (ii) *Grid services*, in our case it is provided by Globus ToolKit [4]; (iii) *High level tools*, like brokers or meta-schedulers; (iv) *Grid applications*, where an API specification, such as SAGA [5] or DRMAA [6], is required to ease the application development.

The Globus toolkit [4], *de facto* standard in Grid technology, provides the services and libraries needed to enable secure multiple domain operation with different resource management systems and access policies. Globus constitutes the Grid service layer, as it supports the submission of applications to remote hosts by providing resource discovery, resource monitoring, resource allocation, and job control services.

Probably one of the most important high level tools in a computational Grid is the application scheduler. In this research we will use the Grid*W*ay framework [7]. Grid*W*ay is a light-weight meta-scheduler that performs job execution management and resource brokering. It allows unattended, reliable, and efficient execution of jobs on heterogeneous and dynamic Grids, see [7] for a detailed description.

Finally, MPICH-G2 [8] and DRMAA [6] are the most used programming paradigms to develop Grid applications. MPICH-G2 is a complete implementation of MPI-1 standard that uses the Globus Toolkit to support Grid operations. MPICH-G2 has been previously

Scheme	Chunk size (C_i)	
CSS(2000)	2000 2000 2000 2000 2000 2000 2000 2000 2000 2000 2000	...
GSS(1000)	10000 8000 6400 5120 4096 3277 2622 2097 1678 1342 1074	...
TSS(5000,1000)	5000 4750 4500 4250 4000 3750 3500 3250 3000 2750 2500	...
FISS	2000 2000 2000 2000 2000 3334 3334 3334 3334 3334 4668	...

Table 1. Sample chunk sizes with 5 nodes for the CSS, GSS(min.chunk), TSS(first, last), FSS and FISS algorithms.

used to develop several Grid applications [9, 10, 11]. These applications are generally not scalable, nor reliable and do not allow sharing of widely separated resources. On the other hand, DRMAA is an API specification to different DRMS (Distributed Resource Management Systems), in our case Grid*W*ay [12], and allows to manage job submission, monitoring and control and retrieval of job status. Moreover, DRMAA allows to develop loosely-coupled applications (defined in Section 4) as DRMAA delegates to the DRMS the responsibility of job execution, fault tolerance and migration.

4 Loosely-coupled loop scheduling

MPICH-G2 have been previously used to develop the self-scheduling loops applications in a Grid environment [3]. However, these applications have three main disadvantages, namely: all resources must be simultaneously allocated to begin execution of the application, due to an implicit DUROC barrier [8]; it is necessary to restart the self-scheduling loop when a resource fails during the execution; and it is no possible to join new resources to a currently running application. We will refer these applications as tightly-coupled implementations.

This work presents the development of a loosely-coupled approach, that implements master-worker schema to develop self-scheduling loops. In particular, we propose a new implementation of these algorithms that presents the following characteristics:

- *Reliability.* When a resource fails, the execution of the whole application continues, because the worker jobs will be restarted in other resource.
- *Dynamic Adaptation.* The worker loops can migrate to more suitable resources during the execution of the application. Furthermore, during the execution of loosely-coupled self-scheduling loops new resources can be used to execute the remainder worker loops.
- *Transparency.* The worker loop execution, fault tolerance capabilities and worker loop migration are transparent from the developer point of view, as the implementation with DRMAA delegates this tasks to the DRMS.

However, the main disadvantage of the DRMAA approach, compared to the MPICH-G2 socket connections, is the need for storing the partial results in secondary storage to distribute these results over Grid resources. This fact may increase the total execution time in some situations.

Figure 1 shows the method used to implement a loosely-coupled loop scheduling. Let us considerer two $N \times M$ matrixes, B and C, and their addition A. The translation from the sequential implementation (diagram 1 of Figure 1) to a loosely-coupled loop scheduling program (diagram 2 and 3), consists of: (i) an executable that implements the worker loops and stores the partial results in a file (diagram 2); (ii) a master loop using a specific self-scheduling scheme that distributes worker loops over the Grid with DRMAA sentences (diagram 3). Finally, this master loop waits for the end of the worker jobs execution.

5 Experimental Results

In this section we analyze the execution of the loosely-coupled loop scheduling algorithms described previously. In particular, we consider an application that solves the Mandelbrot set problem [13] on the domain $[-1.7, 0.8] \times [-1.0, 1.0]$ for a window size 60000×50000 pixels with 6 bits per pixel (2.1 Gigabytes). In these experiments, the window is divided in different horizontal stripes. The size of each stripe is defined by the self-scheduling algorithm, with $size_i = 60000 \times C_i$. The experiments were conducted in the UCM research testbed, based on the pre-WS services of the Globus Toolkit 4 [4], briefly described in Table 2. These machines are slightly heterogeneous to prove the functionality of self-scheduling methods

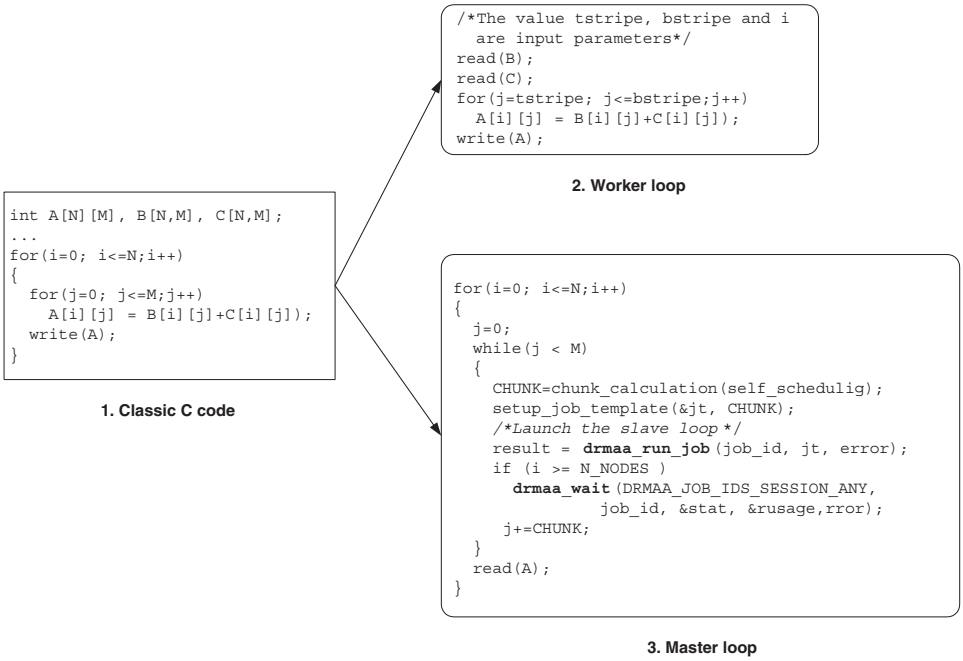

Figure 1. Translation of a classic C loop to a loosely-coupled loop.

and isolate the effects caused by highly heterogeneous systems.

The Mandelbrot set implementation is divided in two tasks. The first task (master process) calculates the chunk size depending of the self-scheduling scheme and launch the worker process over the Grid. The second task, the worker process, solves the Mandelbrot set on a given stripe and returns the results to the master process. The distribution on the worker processes is not uniform, as: the chunk size differs in each node and iteration (with the exception of the CSS scheduler) see Table 1; and the computational workload of each loop iteration is not uniform.

Table 3 shows the average CPU time ($\overline{T_{cpu}}$), file transfer time ($\overline{T_{xfr}}$), wait time in the PBS queue ($\overline{T_{queue}}$) and the resource utilization (U_i) in the execution of six experiment sets. The resource utilization is defined as:

$$U_r = \frac{T_r^{cpu}}{T_{wall}} \text{ where } r \in \{\text{hydrus}, \text{ursa}, \text{cygnus}, \text{draco}\},$$

where T_r^{cpu} is the total CPU time of all the worker loops executed in resource r, and T_{wall} is the wall time of the experiment. Note also that T_r^{cpu} does not include the wait time (e.g. in a PBS queue) and middleware overheads. As can be seen, resource utilization ranges from 70% to 90% in most of the experiments. However, for 7 nodes and CSS algorithm the resource utilization of hydrus decreases dramatically. This is due to the saturation of PBS system, note that the PBS queue time is 77% of the worker CPU time. Moreover, when the number of worker jobs is similar to the number of nodes, and it is not multiple of it, the resource utilization of some nodes also decreases (see Table 3 with machine cygnus, GSS algorithm and 7 nodes).

The effects of previous consideration can be clearly seen if we consider the speed-up of the application (Figure 2).

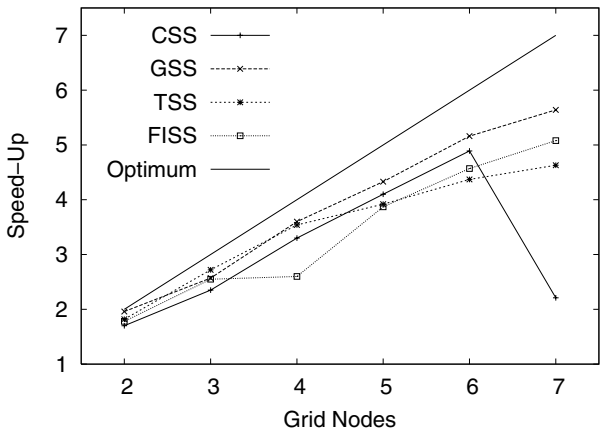

Figure 2. Speed-Up of the loosely-coupled loop scheduling Algorithms.

Name	Model	OS	Speed	Memory	Job Mgr.
hydrus	4×Intel P4	Linux 2.6	3.2GHz	512MB	PBS
ursa	Intel P4	Linux 2.6	3.2GHz	512MB	fork
cygnus	Intel P4	Linux 2.6	2.5GHz	512MB	fork
draco	Intel P4	Linux 2.6	3.2GHz	512MB	fork

Table 2. Characteristics of the machines in the UCM research testbed.

CPU	HOST	CSS	GSS	TSS	FISS
2	hydrus	259/24/9/82%	1017/24/2/96%	375/25/3/87%	1118/24/2/96%
	draco	243/22/-/86%	747/22/-/94%	385/22/-/90%	890/22/-/76%
3	hydrus (x2)	243/24/2/86%	410/23/2/76%	405/24/2/88%	668/23/2/82%
	draco	254/22/-/86%	2364/22/-/98%	338/22/-/89%	678/22/-/83%
4	hydrus (x3)	266/24/6/83%	415/24/3/88%	393/24/2/89%	550/24/2/68%
	draco	230/23/-/85%	818/23/-/95%	397/23/-/90%	525/23/-/66%
5	hydrus (x4)	259/25/3/84%	418/24/2/87%	432/24/3/81%	400/25/3/75%
	draco	253/22/-/83%	310/22/-/87%	303/22/-/77%	493/24/-/92%
6	hydrus (x4)	247/25/5/81%	430/26/2/89%	419/38/5/73%	370/24/3/72%
	draco	260/23/-/82%	255/23/-/86%	412/31/-/85%	414/22/-/91%
	ursa	265/23/-/83%	249/24/-/82%	320/32/-/69%	374/24/-/83%
7	hydrus (x4)	1125/28/870/18%	320/27/3/81%	462/25/3/72%	294/26/3/73%
	draco	254/23/-/54%	309/25/-/79%	245/24/-/68%	396/25/-/91%
	ursa	322/23/-/57%	463/23/-/80%	1387/24/-/97%	287/26/-/66%
	cygnus	259/23/-/55%	321/27/-/55%	307/26/-/43%	438/27/-/67%

Table 3. Execution results of the Mandelbrot set problem with different number of nodes and self-scheduling algorithms: $\overline{T_{cpu}}$ / $\overline{T_{xfr}}$ / $\overline{T_{queue}}$ (sec) / U_i.

In general, the behavior of algorithm is correctly adapted to the ideal speed-up, except for CSS (PBS problem) and TSS (wrong chunk distribution of the algorithm) self-scheduling schemes with 7 nodes. During the experiments, the fault rate was 2%. We will like to remark that in these cases has not been necessary to restart the whole application, what would have increased the total CPU time.

6 Conclusions

In this work, we have presented the implementation of loosely-coupled loop scheduling algorithms in a Grid environment based on DRMAA standard. The main advantages of this approach, in comparison with the tightly-coupled implementations (eg. MPICH-G2) are: *reliability, dynamic adaptation* and *transparency*. We have also demonstrated how Grid characteristics, like fault rate and local resource management system queue time, can degrade the execution time. This factor has not been previously considered to develop dynamic scheduling algorithms.

References

[1] Yang, C.T., Chang, S.C.: A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters. In: Proc. of the 2002 Inter. Computer Symposium. (2002)

[2] Chronopoulos, A.T., Andonie, R.: A Class of Loop Self-Scheduling for Heterogeneus Clusters. In: Proc. 2001 IEE Int. Conference on Cluster Computing. (2002)

[3] Cheng, K.W., Yang, C.T., Lai, C.L., Chang, S.C.: A Parallel Loop Self-Scheduling on Grid Computer Environments. In: Proc. of the 7th Int. Symp. Parallel Architectures, Algorithms and Networks. (2004)

[4] Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. International Jour-

nal of Supercomputer Applications **11** (1997) 115–128

[5] Merzky, A.: SAGA Strawman API. Technical report (2005)

[6] Rajic, H., et al.: Distributed Resource Management Application API Specification 1.0. Technical report, DRMAA Working Group – The Global Grid Forum (2004)

[7] Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. J. Software – Practice and Experience (2004)

[8] Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Parallel and Distributed Computing **63** (2003) 551–563

[9] Chen, J., Taylor, V.: Mesh Partitioning for Distributed Systems. In: 7th IEEE Symp. on High Performance Distributed Computing. (1998)

[10] Larsson, O.: Implementation and Performance Analysis of High-Order CEM Algorithm in Parallel and Distributed Environment. Technical report (1998)

[11] Mahinthakumar, G.K., Hoffman, F.M., Hargrove, W.W., Karonis, N.T.: Multivariate Geographic Clustering in a Metacomputing Environment Using Globus. (1999)

[12] Herrera, J., Huedo, E., Montero, R.S., Llorente, I.M.: Developing Grid-Aware Applications with DRMAA on Globus-based Grids. Lecture Notes in Computer Science **3149** (2004) 429–435

[13] Mandelbrot, B.B.: Fractal Geometry of Nature. W.H. Freeman & Co. (1988)