Applying Backfilling Over a Non-Dedicated Cluster

Dept. Informática
Universitat Autónoma de Barcelona, Spain
{p.hernandez, e.luque}@cc.uab.es, mauricio@aomail.uab.es

Dept. Informática
Universitat de Lleida, Spain
{sisco,francesc}@eps.udl.es

Abstract

The resource utilization level in open laboratories of several universities has been shown to be very low. Our aim is to take advantage of those idle resources for parallel computation without disturbing the local load. In order to provide a system that lets us execute parallel applications in such a non-dedicated cluster, we use an integral scheduling system that considers both Space and Time sharing concerns. For dealing with the Time Sharing (TS) aspect, we use a technique based on the communication-driven coscheduling principle. This kind of TS system has some implications on the Space Sharing (SS) system, that force us to modify the way job scheduling is traditionally done. In this paper, we analyze the relation between the TS and the SS systems in a non-dedicated cluster. As a consequence of this analysis, we propose a new technique, termed 3DBackfilling. This proposal implements the well known SS technique of backfilling, but applied to an environment with a MultiProgramming Level (MPL) of the parallel applications that is greater than one. Besides, 3DBackfilling considers the requirements of the local workload running on each node.
Our proposal was evaluated in a PVM/MPI Linux cluster, and it was compared with several more traditional SS policies applied to non-dedicated environments.

Keywords: Space-Sharing, Non-Dedicated clusters, Backfilling, Coscheduling, Distributed Systems, Cluster Processing, Load Balancing.

1. Introduction

Studies like [1, 2] indicate that the workstations in a NOW are under-loaded, and hence, some resources are wasted. Some past studies [3, 2] demonstrated that it is possible to use those idle resources for parallel computation, generating advantages for the parallel user but not disturbing the local tasks. Many alternatives had been proposed for dealing with such a non-dedicated environment, like remote execution, migration, load balancing, hibernable computers, etc. Our proposal is oriented toward the Job Scheduling [4] alternative.
Parallel job scheduling in a non-dedicated cluster can be performed at two different levels, space and time sharing. Time Sharing scheduling deals with the problem of distributing the CPU time between the parallel and local tasks. TS is done in such a way that it reduces the parallel application execution time by reducing the communication waiting time. This goal is achieved by a technique known as coscheduling [5, 1, 6]. This schema can be done in several ways, however, there are two mayor alternatives. In the first approach, all the processes forming a job are scheduled and descheduled by means of a global context switch. This technique, termed explicit coscheduling, or more generally Gang Scheduling [7, 2, 8], is suitable for environments where the time quantum given to the jobs are large enough to justify the global context switch. The second alternative relies primarily on local communications events (arrival and/or waiting for a message), to determine when and which process to schedule. A technique based on this alternative is Cooperating CoScheduling [1] (CCS), developed by our group and extensively evaluated in the past [9, 3]. It is based on increasing the receiving task priority, even causing CPU preemption of the task being executed. This kind of technique is more suitable for smaller quantums, and hence, for environments with some interactive local load running. Therefore, this technique fits into the requirements of a non-dedicated cluster. In addition, CCS provides some load balancing characteristics and a job interaction mechanism. The load balancing schema tries to uniformize the resources given to each task of a parallel job. On the other hand, the job interaction mechanism lets the system control the level of intrusion into the local workload.

The other aspect to consider for doing job scheduling is the Space Sharing (SS) concern. According to the TS system characteristics explained above, the traditional SS policies have to be modified in order to be applied in a non-dedicated environment. This leads us to divide the SS policies into three different classes, where each one solves one different scheduling problem. The first one faces the problem of selecting the best set of nodes for executing an application (Node Selection policies), considering a non-dedicated cluster and its state. The second set of policies deals with the Job Selection process (i.e. Backfilling, Best Fit, Just First, etc) from a waiting queue, while the third set deals with the Job Ordering or prioritization process (i.e. First Come First Serve - FCFS, Shortest Job First - SJF, Largest Job First - LJF, etc).

Regarding to the Node Selection problem, very little work (if any) has been done to study the effects of a communication-driven coscheduling system over the SS schema, and even less, if we consider non-dedicated clusters. In this work, we have defined some new approaches, termed Normal and Uniform, applicable to this kind of environment. Both policies consider two different variables for assuming scheduling decisions: the cluster state (intrusion level into the local workload, the parallel applications MPL and the memory and CPU usage) and the available nodes.

On the other hand, the most successful approach described in the literature for dealing with this Job Selection and Job Ordering problem, is the combination of a simple FCFS queue with a Backfilling [10, 7] technique. With this kind of policy, a queued application not at the head could be executed, whenever this does not delay the start of the queued job at the head (EASY Backfilling [10]). Thus, backfilling requires the future state to be estimated in order to know when running jobs will finish and free up their processors.

In this paper, we analyze the combination of our defined Node Selection policies with a FCFS-Backfilling schema. It means that the two variables stated by the node selection policies (cluster state and available nodes), are complemented with the future estimation carried out by the backfilling policy. This determines a three dimensional policy, termed by us 3DBackfilling.

This policy was compared with a set of traditional SS scheduling policies. The evaluation was carried out using an integral job scheduling system developed by us, termed CISNE [6]. The analysis shows the good performance and applicability of our proposal for this kind of environment.

The remainder of this paper is as follows: in section 2 we define our problems. In section 3 the proposals for solving the stated problems are described. The efficiency measurements of our policies are performed in section 4. Finally, the main conclusions and future work are explained in section 5.

2. SS and TS Interaction Problems

In the past, several efforts [4], were applied for scheduling parallel application in dedicated clusters. In such an environment, the whole load was controlled by a global scheduling system that considered both Time and Space sharing concerns.

Considering the TS problem, Figure 1 shows a taxonomy used for classifying several coscheduling methods. It is important to note that most of the past efforts were focused on explicitly coscheduled clusters (i.e. Gang Scheduling - GS [5] in Figure 1), where the coscheduling is guaranteed by a global context switch. In such a system, the parallel machine could be seen as a set of n parallel virtual machines (VM), forming what is known as an Ousterhout matrix [11]. The matrix provides information about the parallel jobs and their forming tasks, as well as mapping onto the VMs. Every VM is synchronized to the others by means of a global context switch. Thus, there is no interaction among the VMs, which also means none between the parallel tasks running in the same node. Gang scheduling, is an efficient coscheduling algorithm, which has mainly been used in supercomputers (CM-5, SGI workstations and so on). The advantage of a gang scheduler is a faster completion time, because the processes of a job are scheduled together. Its disadvantage is the global synchronization overhead needed to coordinate a set of processes and the high overhead introduced to local tasks. A GS coscheduling system normally uses quantums that are too large for providing interaction.

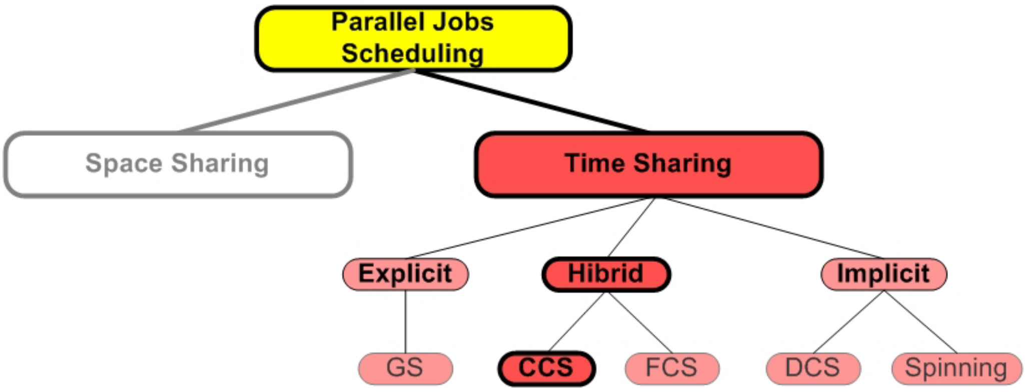

Figure 1 Time Sharing Taxonomy.

A more suitable alternative was proposed for non-dedicated clusters, where the quantum should be short enough for executing interactive local tasks. This kind of system, termed implicit, uses the communication behaviour of any parallel application for coscheduling. Among the proposed policies we can find the Demand based CoScheduling (DCS [12] in Figure 1) that modifies the parallel task priority on behalf of its communication behaviour, even causing CPU preemption of the current running task. Another alternative relies on Spinning [13] in the CPU after sending a message, waiting for a response to it. Finally, it is possible to find some hybrid approximations that combines characteristics from both implicit and explicit systems. One proposal, called Flexible CoScheduling (FCS [14] in Figure 1) applies GS to certain applications on behalf of its communication pattern (as an implicit method does).

On the other hand, our TS system termed CCS [1] is based on a DCS system for managing the coscheduling, but enhanced with some explicit information exchange for balancing the resources given to the parallel tasks, while preserving the local task responsiveness. However, and due to CCS's particular characteristics, we came across some problems applying traditional SS techniques to a non-dedicated cluster managed by it.

The main problem faced is related to the lack of an Ousterhout matrix, present in every explicit coscheduling schema. This is illustrated in Figure 2, that shows the cluster state for an explicit (a) and an implicit (b) TS system. In both cases, there is some load already running (present load), while a couple of jobs (J_j and J_k) are launched. Figure 2.a shows how J_i and J_k run on their own VMs (i.e. matrix rows 3 and 2 respectively) with no interaction with the other present load in the system. This behaviour assures the same computational power for every task of any job, and a MultiProgramming Level (MPL) of at most one application running on each VM (MPL = 1). Besides, it is possible to apply any SS technique to each VM, and consider an MPL of at most one.

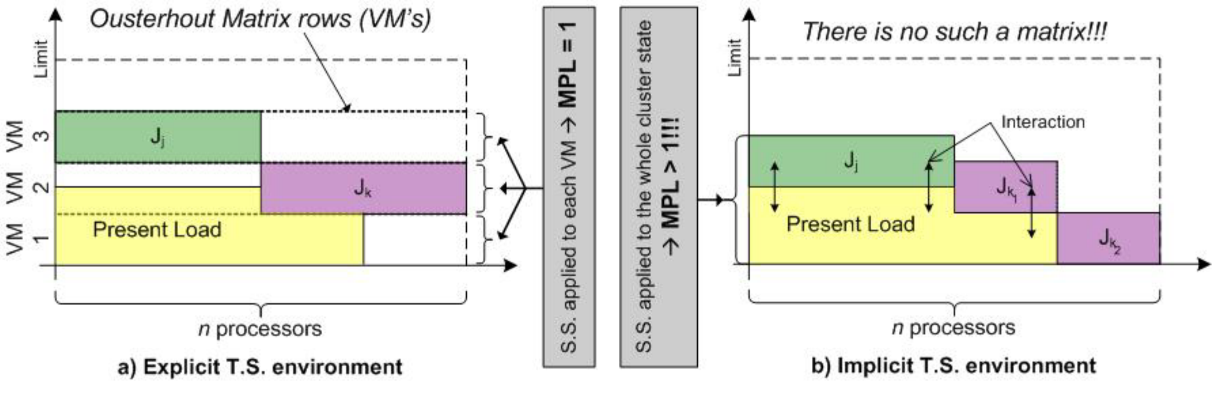
Ousterhout Matrix rows (VM's)
Limit
VM 3
VM 2
VM 1
J_j
J_k
Present Load
n processors
a) Explicit T.S. environment
S.S. applied to each VM → MPL = 1
S.S. applied to the whole cluster state → MPL > 1!!!
There is no such a matrix!!!
Limit
Interaction
J_j
J_{k_1}
J_{k_2}
Present Load
n processors
b) Implicit T.S. environment

Figure 2 Difference between applying SS to an explicit and dynamic (implicit) TS environment.

In implicit coscheduling in general, and in CCS in particular, there is no such a matrix. Thus, we have only one VM, the whole cluster state, to which apply our SS techniques apply. As a consequence, the dimensionality of the SS problem is incremented because we have more than one application running on our single VM in the same set of nodes. This means a dynamic MPL greater than one (MPL > 1) across the cluster. Moreover, due to a lack of a global context switch, the tasks running in the same node have to compete for the CPU. Therefore, the computational power given to each task of a parallel job depends on the load of each node. This computational power difference among a job's tasks, makes job distribution a really important problem, because the coscheduling performance could be affected by the job placement.

Figure 2.b depicts how J_j and J_k interact with the other present load (which could be either parallel or local). An example of different computational power could be seen for the J_k case, where some of its tasks have to share its CPUs with some load while others should not.

3. 3DBackfilling Proposal

In order to tackle the problems stated above, we first define a taxonomy that lets us fix our proposals among the existing SS policies, and how our approach deals with a non-dedicated environment. Based on this, we merge policies from different classes taken from the taxonomy, to propose a SS approach applied to TS systems based on implicit coscheduling techniques and oriented to non-dedicated clusters.

3.1 Our Space Sharing Taxonomy

Even considering that there are some idle resources usable for parallel computation, these are finite. In consequence, a job cannot be executed every time it arrives at the system. In those cases where a job could not be executed, it has to wait in a jobs waiting queue. Considering such condition we have to manage those available resources and tackle several SS problems. Such problems include selecting the best set of nodes for executing a given job for a given cluster state, or the definition of a process for selecting a job for execution from the waiting queue or the relative priorities (order) between those waiting jobs. To deal with these concerning, we define a taxonomy (Figure 3) that let us face those problems separately, while it is possible to merge the solutions for each problem in a complete scheduling policy.

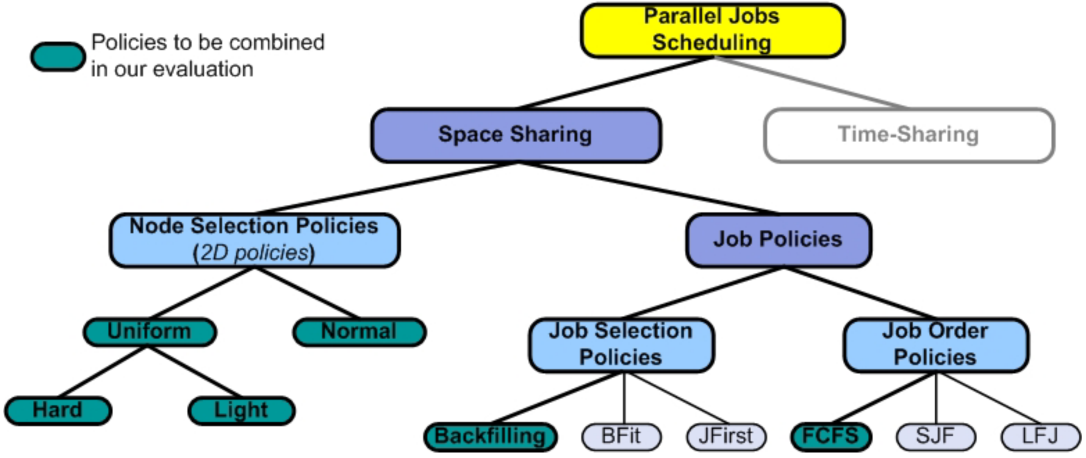

Figure 3 Space Sharing Taxonomy.

Assuming our taxonomy, we firstly focus on the class of policies termed *Node Selection*, which are oriented towards the selection of the best subset of nodes from the cluster to execute a given application. This process is done considering two main variables, the available nodes and its current state. For this reason, we have also called this kind of policy, 2D policies. It should be noticed that the defined Node Selection policies are mainly aimed at helping the coscheduling mechanism to perform better, and hence, to minimize the applications execution time.

The first idea for enhancing coscheduling performance is oriented towards controlling the level of resources given to the parallel tasks throughout the cluster.

This is done by not overloading a node with some local or parallel load already running. Following this principle, we propose a policy termed *Normal* that limits the resources used by the parallel applications across the cluster. The Normal policy launches an application on any set of nodes where the fact of executing it does not mean exceeding a system usage limit for some resource. This acceptable limit is established by the means of a social contract [15], and sets up the maximum parallel MPL or the percentage of memory or CPU that can be used by the parallel applications on each node. Besides, in order to select the best subset, this policy implements a mechanism that gives priority to the candidate nodes for executing an application. This prioritization mechanism is managed by parameters like the CPU or Memory usage in a node by the local and parallel load. It should be noticed, that in this prioritization process the most important parameter is the memory used in each node, followed by the MPL and CPU usage. This is justified because we want to avoid overloading the memory (i.e. avoiding paging), which is the worst problem that the TS has to face.

Nevertheless, using the Normal policy we are still not considering the load interaction inside a node (Figure 2.b). Therefore, we add new characteristics to the scheduling decision process carried out by the Normal policy, defining a new policy, termed *Uniform*. This policy is characterized by the following: (a) it executes tasks from differently oriented applications (i.e. communication or computation bound) in the same node and (b) it runs applications one over another in an ordered manner, whenever possible. By ordering the applications we mean launching parallel applications in such a way that a couple of parallel applications run in the same set of nodes, trying to uniformize the load in every node given to a job. Figure 4.a shows how the Uniform policy executes a CPU bound application (J_3) in the same set of nodes as a communication bound application (J_2). In contrast, in fig. Figure 4.b a Normal policy executes the J_3 application regardless of the load and its orientation. In this case, the computational power assigned to J_3 is not the same for all of its tasks. However, in both cases the established system limit for every selected parameter is preserved by the defined policies.

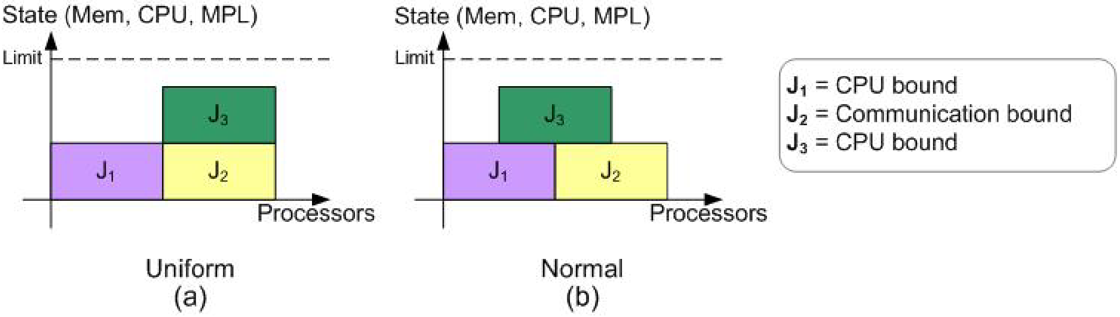

Figure 4 Scheduling difference between the Normal and Uniform policies.

It should be noticed that the Uniform policy could be used in two ways. The first possible usage imposes the (a) and (b) restrictions in a hard way, which means that if a set of nodes with the desired conditions is not found, the application has to wait in a queue. This usage is called Hard Uniform. On the other hand, it is possible to flexibilize the Uniform rules and launch an application in a Normal way if we do not found a Uniform set of nodes for a given application. In this case, we call the policy Light Uniform.

The Node Selection problem described above only faced the job distribution problem. However some other aspects related to the Job policies (Figure 3) should be taken into account. Based on these, two main concerns have to be faced: the job order in the waiting queue and the job selection process for choosing one job from that queue. These kind of selection and ordering process is what we call more traditional space scheduling policies in dedicated environments [7].

Among the traditional *Job Selection* policies (Figure 3) we have the Best Fit (BFit) approach, that always looks for the job that minimizes the resources left unused. Another strategy, defined by us, is the Just First (JFirst) policy, that being less restrictive than BFit, only tries to execute the first job in the queue. A separation should be made for the Backfilling [10, 7] policy, because it tries to execute applications by estimating the future resource usage. The most extensively analyzed backfilling strategy is the EASY [10] technique, and it states that any queued job can be executed given the fact that it will not delay the start of the job at the front of the queue. To guarantee this restriction, an estimation of the future cluster state has to be made. In principle, this estimation is based on information provided by the user about the execution time of the applications under certain conditions (i.e. number of needed processors and CPU or memory consumption, etc.). Figure 5.a shows how for a given queue, task 5 has to wait until task 3 finishes in order to start its execution. With a Backfilling approach (Figure 5.b), task 5 could be scheduled before task 1 finishes, given the fact that it is expected to finish by task 3 start time.

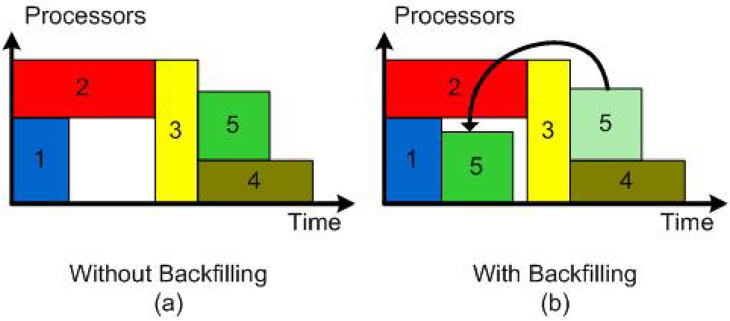

Figure 5 Example of a Backfilling policy.

In addition, every time that a job arrives in the system, it has to be compared to the jobs waiting in the queue. This comparison process lets the system establish if the arrived job has enough priority to be executed at the current moment. Therefore, and to have a complete scheduling mechanism, it is necessary to choose a Job Order policy (Figure 3) that lets us determine the relative priorities (i.e. order) of the jobs in the queue. Among the traditional policies we found First-Come-First-Serve (FCFS), which orders the jobs according to their arrival time or SJF (LJF), that orders the applications in increasing (decreasing) execution time estimation. In order to minimize the number of different sets of merged policies to be evaluated, we fix this class of policy to FCFS for every evaluated schema. The FCFS choice is justified by the fact that most of the backfilling approaches use it as their job ordering policy. This is due to the simplicity, fairness and absence of starvation of FCFS.

Finally, it should be noted that the Job Selection process is traditionally done by trying to maximize the resource usage, and hence, diminishing the job waiting time. By combining policies from the Node Selection and Job Selection classes, the turnaround time of the jobs is diminished in two different metrics, the execution and the waiting time. The turnaround minimization is important for us, due to the characteristics of our environment, where it is important to present some benefits for the parallel user using a non-dedicated cluster. Hence, doing this merging we not only have a complete scheduling proposal, but a scheduling policy that minimizes an important metric (for us) from different points of view.

3.2 Merging Node Selection And Job Policies

In order to have a complete scheduling policy that considers every concerning stated in the previous subsection, it is necessary to select a policy from each defined class (i.e. Node Selection, Job Selection and Job Ordering in Figure 3) and merge them into a complete policy.

Considering that the Backfilling policy has been demonstrated to be one of the most effective job selection approaches, it is desirable for us to merge it into our complete scheduling proposal. However, some concerns have to be taken into account for combining a Backfilling technique with our 2D policies (i.e. Node Selection policies). The main problem comes from the need to estimate the future state of the environment. Such estimation should be applied to the whole cluster state, and not to a VM with an MPL of at most 1 (as in an explicit coscheduled system). Moreover, the estimation process has to consider the local load. Hence, to take a combined scheduling decision (i.e. Backfilling + 2D policy), we have to consider three variables: the available cluster nodes and their state (from 2D policies), and the future cluster state (from the Backfilling technique). The addition of this new concern (i.e. variable), raises our problem dimensionality from 2D to 3D. In Figure 6, it is possible to observe the variables to take into account for assuming a backfilling strategy in our environment. This figure also depicts a decision problem (assumes that a job J arrives at the queue): which is the best thing to do, execute the J application right now in a Normal (J_i) way or wait for a while, and execute it in a Uniform way (J_k)?. This is another problem that arises using a backfilling approach combined with our 2D policies that we have to deal with.

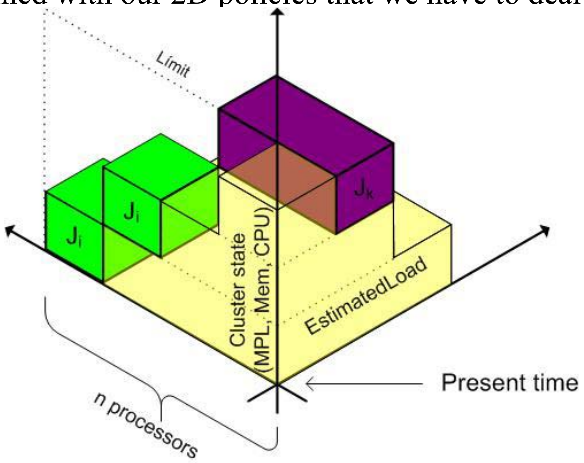

Figure 6 A 3DBackfilling example.

Consequently, we define a pair of policies that include a backfilling mechanism and our 2D policies, termed 3DBackfilling techniques. The first 3D policy, called 3DBackfilling-Hard, is the combination of a backfilling schema with our 2D Uniform-Hard policy (J_k case in Figure 6). In the same way, we defined another policy, termed 3DBackfilling-Light, set up using a Uniform-Light policy (J_i case in Figure 6). The merging of a Backfilling technique with the Normal 2D policy is not considered because it is already included in the Uniform-Light approach.

4. Experimentation

In the present section, we aim to show the evaluation of our SS proposals for dynamically coscheduled, non-dedicated clusters. To carry out the experimentation, we introduce the exercised workloads and metrics. Then, in the second subsection, we present a set of results that shows how our defined SS policies perform.

4.1 Workloads and Metrics

A necessary element for carrying out our evaluation is a way to represent a non-dedicated cluster. On one hand, we need several parallel applications that arrive at some intervals and, on the other hand, we need some local user activity. We also define some other policies to compare our proposal, giving a lower limit to compare with (e.g. considering an MPL = 1). Finally, we measure the system performance using system and user metrics from the parallel and local user point of view.

The parallel workload was a list of 90 PVM/MPI NAS parallel applications with a size of 2, 4 or 8 tasks that reached the system following a Poisson distribution, described in [5]. The chosen NAS applications and their resource consumptions are depicted in Table 1.

	Memory (min/max)	CPU (min/max)	Exec. Time (min/max)
CG	55 / 120 MB	67 / 75 %	37 / 51 sec.
IS	70 /260 MB	58 / 69 %	40 / 205 sec.
MG	60 / 220 MB	82 / 89 %	26 / 240 sec.
BT	7 / 60 MB	85 / 93 %	90 / 180 sec.

Table 1 The NAS benchmarks used.

The parallel applications were merged so that the entire workload had a near-balanced requirement of computation and communication: each application comprised approximately 25% of the workload. It is important to note that the MPL reached for the workload depended on the system state at each moment, but in no case surpassed an MPL = 4. This was established in order to respect the social contract, which was set at 50% of the resources available for each kind of load (local/parallel) [3]. Besides, the system uses the whole node for the parallel applications if there is no local load.

Moreover, to evaluate the influence of the relation between the application execution time and the application inter-arrival time, we define two type of workloads. The first one, termed SIT (Simple Inter-arrival Time), has an application inter-arrival time shorter than the average application execution time. The second, named DIT (Double Inter-arrival Time) makes the inter-arrival of the same order as the execution time. The case for an inter-arrival time greater than the execution time is not treated because in this scenario, the SS policy is almost irrelevant considering that every time an application arrives it finds the cluster idle.

On the other hand, the local workload was carried out by running a synthetic benchmark. This allowed the CPU load, memory requirements and network traffic used by the local user to be fixed. To assign realistically these values, we monitored the average resources used by real users. According to this monitoring, we defined two local user profiles. The first profile identifies 65% of the users with high inter-activeness needs (called XWindows user: 15% CPU, 35% Mem., 0,5KB/sec LAN), while the other profile distinguishes 35% of the users with web navigation needs (called Internet user: 20% CPU, 60% Mem., 3KB/sec. LAN). This benchmark alternates CPU activity with interactivity by running several system calls and different data transfers to memory. In order to measure the level of intrusion into the local load, our benchmark provide us with the system call latency. Besides, and according to the monitorized values, we loaded 25% of the nodes with local workload in our experiments.

Both workloads were executed in a Linux cluster using 16 P-IV (1,8GHz) nodes with 512MB of memory and a fast ethernet interconnection network. In addition, a job scheduling system developed by us, termed CISNE [6, 16], was used to apply our proposals. This system integrates our TS system, CCS, with a job scheduler that lets us implement our 2D and 3D proposals easily.

To give a lower limit to compare our proposals (Normal, Uniform-Hard, Uniform-Light, 3DBackfilling-Hard and 3DBackfilling-Light), we defined an extra pair of policies. The first one, named Basic, is a Normal policy where the maximum MPL is set to 1. This policy will give us an idea of the benefits obtained by multiprogramming the cluster, when compared with the other policies that use an MPL > 1. In addition, we wanted to evaluate this multiprogramming profits, but from the Backfilling point of view. Hence, we defined another policy, termed 2DBackfilling, that merges a Normal policy with a backfilling schema, but with an MPL = 1. In such a scenario, the MPL is not an important variable, and therefore, the dimensionality of our problem is diminished (i.e. 2D-Backfilling: available nodes and its future state), even though we are considering estimation of the future state of the cluster.

Table 2 shows the whole set of evaluated policies composed of one policy from each of the classes defined in Figure 3, and the allowed MPL in each case.

	Node Selection	**Job Selection**	**Job Order**	**MPL ≤**
Basic	Normal	JFirst	FCFS	1
Normal	Normal	JFirst	FCFS	4
Uniform-Light	Uniform	JFirst	FCFS	4
Uniform-Hard	Uniform	JFirst	FCFS	4
2DBackfilling	Normal	Backfilling	FCFS	1
3DBackfilling-Light	Uniform	Backfilling	FCFS	4
3DBackfilling-Hard	Uniform	Backfilling	FCFS	4

Table 2 The whole set of evaluated policies.

To evaluate our techniques, we first show some results comparing our proposed policies for a plain Linux scheduler and our CCS TS system. This allows us to analyze the influence of the SS policy on the coscheduling mechanism. The measures are done by means of the average execution time of the parallel applications for the SIT and DIT workloads. In a second step, we present results for our proposed SS policies considering the SIT and DIT workloads, but only for CCS. In this case we will show the average application waiting, execution and turnaround time, for each workload. To conclude the evaluation from the parallel point of view, we include some values representing the makespan (i.e. the turnaround of the whole workload). With this metric we evaluate how our SS policies perform from the system point of view.

Finally, and to show that our integral scheduling system does not introduce an excessive load on each node, we use the local benchmark system call latency generated data. This way, it is possible to present results obtained for CCS compared with other coscheduling systems and the Linux plain scheduler.

4.2 Results

In the first part of the experimentation, we show how the proposed 2D and 3D policies could diminish the execution time of the parallel applications. Besides, we aim to depict the influence of CCS over the system performance.

The first effect that it worth mentioning is related to the influence of the 2D policies (NORMAL, UNI-HARD, UNI-LIGHT) on the execution time considering the job inter-arrival time. In Figure 7.left, where the SIT workload imposes a greater pressure on the system, it is possible to observe that a Uniform-Hard policy (UNI-HARD) improves the performance of a Normal policy. On the other hand, when the pressure over the system is lower (i.e. DIT workload, Figure 7.right), the performance of either 2D policy is almost the same. This confirms our assumptions about the importance of the job distribution considering the application execution performance, when the job arrival rate is elevated. Besides, combining a Uniform-Hard policy with a backfilling (3DBF-HARD) schema, it is possible to obtain some gains for both workloads, compared with a 3DBF-LIGHT approach. The Uniform-Hard gains are justified by an enhancement of the coscheduling system performance. This enhancement is obtained because it is easier for CCS to coschedule parallel applications when the tasks running in the same node have different CPU-I/O requirements.

Figure 7 also shows the gains for the coscheduling system. It is important to remark that for the SIT workload, the load imposed on the system is greater due to the short inter-arrival time, which in turn increases the reachable MPL throughout the cluster. Therefore, the execution time for the SIT workload is, on average, higher than in the DIT case. Nevertheless, even considering a higher load, the performance of the CCS system compared to the plain Linux scheduler could reach 15% (3DFB-LIGHT policy) for the SIT workload. On the other hand, with a lower pressure over the waiting job queue (i.e. the DIT workload), and hence, a lower MPL across the cluster, the gains could reach 24% (NORMAL policy). In addition, for policies with an MPL = 1 (Basic and 2DBF), we also observe gains for the CCS system due to an enhancement when there is local user activity.

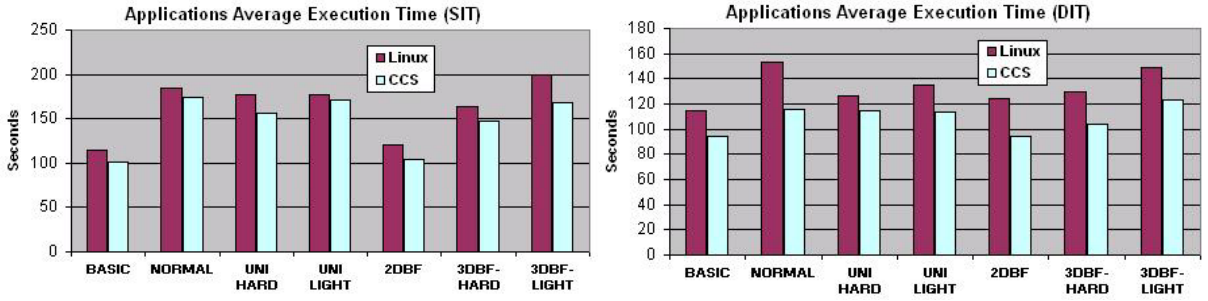

Figure 7 Application execution time for the SIT (left) and DIT (right) workloads.

In the Figure 8 we can observe the wait, execution and turnaround time for the evaluated SS policies under CCS for the SIT and DIT workloads. Figure 8.a shows that due to a short job inter-arrival time, a policy that reduces the waiting time (backfilling) is preferable to another one that reduces the execution time, whenever we are scheduling with an MPL > 1. On the other hand, when the job execution time is similar to the job inter-arrival time (Figure 8.b), it is preferable to diminish the first, and hence, a backfilling (job selection) policy has almost no influence compared with the job allocation policy (Uniform Hard, Uniform Light and Normal policies). The inclusion of a backfilling policy is, nevertheless, harmless in this situation, so we can use it in both scenarios.

Therefore, when the waiting time is the predominant factor, we want to schedule applications fast, with little care about the job distribution (Uniform Light or Normal policies). However, when the execution time is more important, we want to schedule well, which means trying to reduce the execution time by helping the coscheduling system with a more intelligent job distribution over the cluster (i.e. 3DBF-Hard).
Considering the influence of the MPL, it is clear that using an MPL greater than one is always preferable over more conservative policies such as Basic or 2DBF that use an MPL = 1. This is because the reduction in the waiting time is greater than the increment in the execution time due to the MPL > 1, which results in a minimization of the turnaround time.

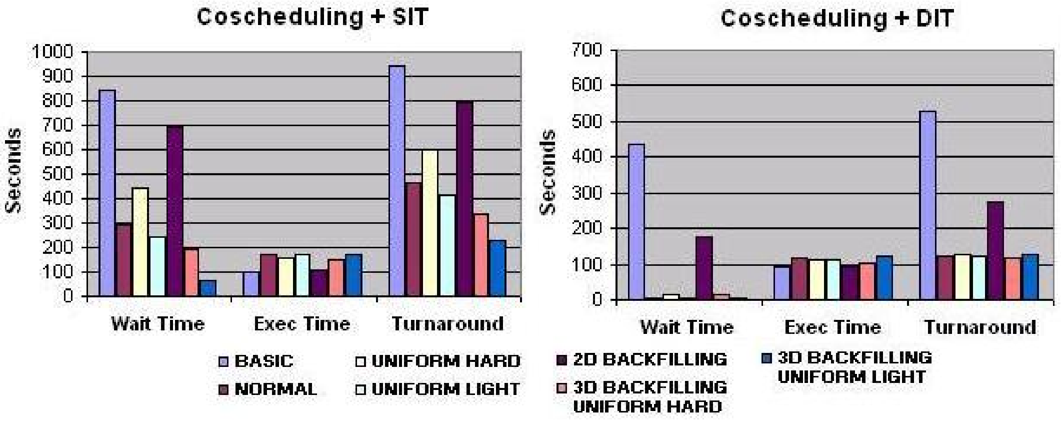

Figure 8 Wait, execution and turnaround time for the evaluated SS policies under CCS.

From the system point of view, Figure 9 presents the makespan for the SIT and DIT workloads. From the figure it is clear that a policy that schedules applications fast (Normal, Uni Light, 3DBF-Hard and 3DBF-Light) is better than another that is more restrictive (Uni Hard or 2DBF). A policy such as Uniform Hard takes the same time for both workloads due to the restrictions imposed. However, if we are using resources that would otherwise be wasted, we believe that this kind of behaviour is tolerable if the user metrics are enhanced, which is the case for workloads like DIT. On the other hand, policies that merge backfilling are very suitable from the makespan point of view. This is true when the job inter-arrival time is short and the restrictions imposed by the node selection policy (Uni Light) are lighter. This scenario is depicted by the 3DBF-Light policy which gives us a really good performance (35% better than Basic) for the SIT workload.

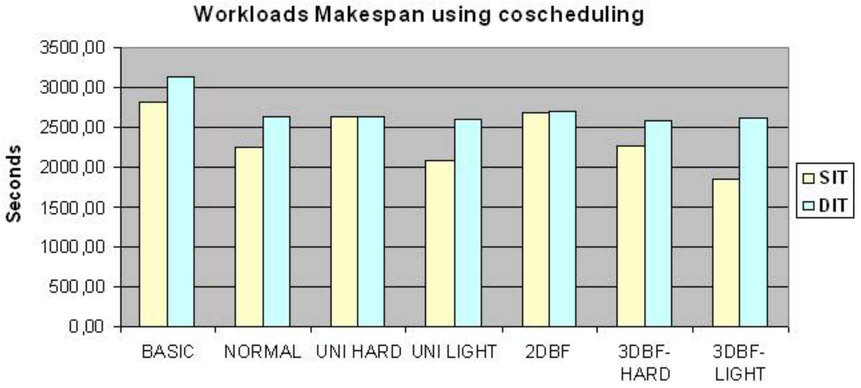

Figure 9 Workloads Makespan.

Finally, we consider that it is fundamental to include some results concerning the system intrusion over the local tasks. Therefore, Figure 9 shows the performance of our CCS system in relation to the plain Linux scheduler and two well known communication-driven coscheduling strategies: Spinning and DCS coscheduling. In implicit coscheduling, a process waiting for messages spins for a determined time before blocking. In contrast, DCS coscheduling deals with all message arrivals (like CCS, but without resource balancing and local jobs preservation). It works by increasing the receiving task priority, even causing CPU preemption of the task being executed inside. Besides, they were evaluated by running the SIT parallel workload for several values of MPL (1 to 4), and applying a 3DBackfilling-Light SS policy. The choice of this SS policy is due to the higher load that it imposes on the system, and hence, a worst situation for the TS system.

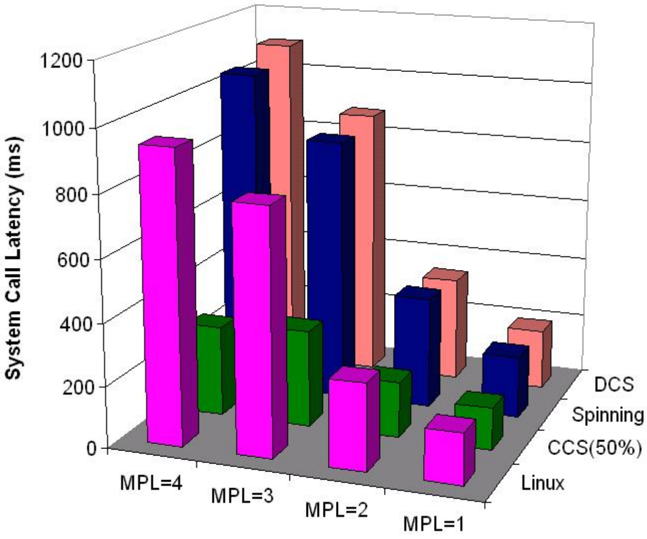

Figure 10. System call latency under the evaluated policies.

In the figure it is possible to observe how the social contract implemented by CCS always maintains the response time (measured by mean of the local benchmark system call latency) under 400ms. This limit for the response time, established by [17, 18], is an acceptable threshold before the user can notice a lack of inter-activeness. Hence, our system really protects the local users from an excessive intrusion of the parallel applications.

5. Conclusions and Future Work

This work presents a set of new SS policies oriented towards dynamically-coscheduled, non-dedicated clusters (2D and 3D policies). Using our policies and an integral scheduling system (Time and Space Sharing system), the paper analyzes how the performance of an implicit coscheduling system could be affected by the distribution policy over a non-dedicated cluster. With this aim, we evaluated our proposed policies, some of them complemented with a backfilling schema. From the combination of our proposals with a backfilling technique, a new backfilling approach for non-dedicated clusters arises. We have called this policy 3DBackfilling, and it is also evaluated in this work. The policies were evaluated using a Linux cluster and considering user and system metrics, from the parallel and local user points of view. We found that a Uniform policy (i.e. a set of applications running on the same set of nodes), can enhance the coscheduling performance compared with other approaches. Nevertheless, in systems with a higher load, it is preferable to reduce the waiting time by combining such policy with a backfilling schema (3DBackfllings). In addition, the inclusion of such a backfilling technique was shown to be very profitable in some cases, while it is never harmful. To resume, when the load is high it is preferable to diminish the waiting time (i.e. 3DBF-LIGHT), while with lower loads it is preferable to diminish the execution time (3DBF-HARD). Doing this we assure the minimization of the turnaround time, which is our main parallel user metric.

Considering our future work we want to increase the system predictability, thus allowing us to establish the turnaround time within a certain range. In order to do this we will include a historical system that lets us estimate some parameters for the executing jobs. Besides, we will study the characterization of the parallel applications and the local user behaviour, and how this could be included into the estimating schema.

References

1. Giné, F. Cooperating Coscheduling: a coscheduling proposal for non-dedicated, multiprogrammed clusters. *PhD Thesis*, U*niversitat Autònoma de Barcelona,* 2004**.**

2. Setia, S., Squillante, M. & Naik, V. The Impact of Job Memory Requirements on Gang-Scheduling Performance 1999.

3. Hanzich, M.; Giné, F.; Hernández, P.; Solsona, F. & Luque, E. Coscheduling and Multiprogramming Level in a Non-dedicated Cluster. EuroPVM/MPI 2004, LNCS, 2004 , 3241, 327-336.

4. Feitelson, D.G.; Rudolph, L.; Schwiegelshohn, U.; Sevcik, K.C. & Wong, P. Theory and Practice in Parallel Job Scheduling Lecture Notes in Computer Science, 1997, 1291, 1-34.

5. Feitelson, D.G. Packing schemes for gang scheduling Job Scheduling Strategies for Parallel Processing, Springer-Verlag, LNCS, 1996, 1162, 89-110.

6. Hanzich, M. Combining Space And Time Sharing On A Non-Dedicated NOW Universitat Autònoma de Barcelona, 2004.

7. Zhang, Y.; Franke, H.; Moreira, J.E. & Sivasubramaniam, A. An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling, and Migration Lecture Notes in Computer Science, 2001 , 2221 , 133-151.

8. Feitelson, D.G. & Jette, M.A. Improved Utilization and Responsiveness with Gang Scheduling Springer Verlag, 1997, 238-261.

9. Giné, F.; Solsona, F.; Hernández, P. & Luque, E. Adjusting Time Slices To Apply Coscheduling Techniques in a Non-Dedicated NOW LNCS, 2002 , 2400 , 234-240.

10. Shmueli, E. & Feitelson, D.G. Backfilling with lookahead to optimize the performance of parallel job scheduling Job Scheduling Strategies for Parallel Processing, LNCS, 2003, 2862, 228-251.

11. Ousterhout, J. Scheduling techniques for concurrent systems Proceedings of the Conference on Distributed Computing Systems, 1982.

12. Sobalvarro, P. & Weihl, W. Demand-based coscheduling of parallel jobs on multiprogrammed multiprocessors Job Scheduling Strategies for Parallel Processing, LNCS, 1995 , 949 , 106-126

13. Anglano, C. A Comparative Evaluation of Implicit Coscheduling Strategies for Networks of Workstations Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00), 2000 , 221-228.

14. Frachtenberg, E.; Feitelson, D.G.; Fernández, J. & Petrini, F. Parallel Job Scheduling Under Dynamic Workloads 9th Workshop on Job Scheduling Strategies for Parallel Processing. HPDC, LNCS, 2003 , 2862 , 208-227.

15. Arpaci, R.; Dusseau, A.; Vahdat, A.; Liu, L.; Anderson, T. & Patterson, D. The Interaction of Parallel and Sequential Workloads on a Network of Workstations ACM SIGMETRICS 1995, 1995, 267-277.

16. Hanzich, M.; Giné, F.; Hernández, P.; Solsona, F. & Luque, E. CISNE: A New Integral Approach for Scheduling Parallel Applications on Non-Dedicated Clusters Accepted at EuroPar 2005, 2005 , to be appear at LNCS.

17. Miller, R. Response Time in Man-Computer Conversational Transactions AFIPS Fall Joint Computer Conference Proceedings, 1968 , 33 , 267-277.

18. Nielsen, J. Nielsen, J. (ed.) Advances in Human-Computer Interaction Intellect Publishers, 1995.