CISNE: A New Integral Approach for Scheduling Parallel Applications on Non-dedicated Clusters*

Mauricio Hanzich[2], Francesc Giné[1], Porfidio Hernández[2], Francesc Solsona[1], and Emilio Luque[2]

[1] Departamento de Informática e Ingeniería Industrial, Universitat de Lleida, Spain
{sisco,francesc}@eup.udl.es
[2] Departamento de Informática, Universitat Autònoma de Barcelona, Spain
mauricio@aows10.uab.es, {porfidio.hernandez,emilio.luque}@uab.es

Abstract. Our main interest is oriented towards keeping both local and parallel jobs together in a non-dedicated cluster. In order to obtain some profits from the parallel applications, it is important to consider *time* and *space* sharing as a mean to enhance the scheduling decisions. In this work, we introduce an integral scheduling system for non-dedicated clusters, termed CISNE. It includes both a previously developed dynamic coscheduling system and a space-sharing job scheduler to make better scheduling decisions than can be made separately. CISNE allows multiple parallel applications to be executed concurrently in a non dedicated Linux cluster with a good performance, as much from the point of view of the local user as that of the parallel application user. This is possible without disturbing the local user and obtaining profits for the parallel user. The good performance of CISNE has been evaluated in a Linux cluster.

1 Introduction

There are several studies in the literature whose main aim is to determine the interaction and effects of space-sharing (S.S.) and time-sharing (T.S.) policies. Nevertheless, most of them are focused on dedicated environments. Furthermore, many of these studies center on Gang Scheduling [1, 2], combined with some kind of backfilling [1] policy for doing the job distribution.

In this work, we want to show a new scheduling approach focused on non-dedicated cluster systems. The use of non-dedicated systems for parallel computation is based on various studies [3] that prove the effectiveness of making good use of the idle CPU cycles by executing distributed applications.

In this article, we present a new system named CISNE. Our system combines S.S. and T.S. scheduling techniques, in order to take advantage of the idle computer resources available across the cluster. CISNE is set up basically of a dynamic coscheduling technique and a job scheduler.

* This work was supported by the MCyT under contract TIC 2001-2592 and partially supported by the Generalitat de Catalunya -Grup de Recerca Consolidat 2001SGR-00218.

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 220–230, 2005.

The dynamic coscheduling system, termed CCS, is the T.S. scheduling component. Traditional dynamic coscheduling techniques [4] rely on the communication behavior of an application, to simultaneously schedule the communicating processes of a job. Unlike those techniques, CCS takes its scheduling decisions from the occurrence of local events, such as communication, memory, Input/Output and CPU, together with foreign events received from remote nodes. This allows CCS to assure the progress of parallel jobs without disturbing local users, and even using an *MultiProgramming Level* (MPL) greater than one. In addition it is possible to re-balance the resources assigned to parallel tasks throughout the cluster. CCS was previously developed [5], and now we present the modifications that allows it to be incorporated into an integral cluster scheduling system, such as CISNE.

The job scheduler, named LoRaS, is the S.S. scheduling component of CISNE. It is responsible for distributing the parallel workload among the cluster nodes. This is performed by taking into account the state of the cluster system, the characteristics of applications already running and those of the waiting jobs. Based on those considerations and the coscheduling restrictions, different techniques for assigning jobs to processors are proposed and evaluated in this article.

CISNE was implemented in a non-dedicated Linux cluster. In this framework, we evaluated the interaction between T.S. and S.S. techniques. This experimentation shows that our proposal obtains better performance than the rest of the evaluated techniques, as much from the point of view of the local user as that of the parallel applications user and the system administrator.

The remainder of this paper is as follows: in section 2 we explain the main problems to solve and our goals for this article. In section 3 the CISNE system is presented. The efficiency measurements of CISNE are performed in Section 4. Finally, the main conclusions and future work are explained in Section 5.

2 T.S. and S.S. Interaction Problems

The choice of a dynamic coscheduler as a T.S. system is based on the fact that this kind of system is better suited to a non-dedicated environment than an explicit (or gang) T.S. coscheduling schema [6]. However, this choice has some implications for the S.S. schema, that force us to develop our own system.

The main effect could be found in the lack of an Ousterhout matrix [7], present in every explicit coscheduling system. In such a system, the parallel machine could be seen as a set of n parallel virtual machines (VM). The matrix provides information about the parallel jobs and their forming tasks, as well as the mapping onto the VMs. Every VM is synchronized to each other by means of a global context switch. Thus, there is no interaction among the VMs, which also means none between the parallel tasks.

On the other hand, in a T.S. system based on dynamic coscheduling techniques, there is no such matrix. Thus, it is not possible to apply the S.S. techniques to each row. As a consequence, and in order to improve the global performance of the system, the (now existing) interaction among the running ap-

plications has to be considered [8]. Therefore, one of the main goals of this work is to find the kind and degree of interaction between the system management components (T.S. and S.S. schemes) to achieve the maximum performance in distributed tasks without damaging the local ones.

The studies carried out by Choi et al. revealed the sensitivity of the implicit coscheduling techniques in relation to the mapping and the execution order of the parallel applications over the cluster (if the MPL is greater than one). In addition, the type of applications (CPU or communication bound) running concurrently over the cluster, and the global system state, can have a great influence on the coscheduling performance. Another factor to take into account in a non-dedicated cluster is the local user activity, which has to be monitorized periodically. The control of those factors allows the S.S. system to schedule better, while it helps the T.S. system to avoid intrusions into the local tasks.

In such a scenario, the best S.S. scheduling of a parallel workload is not obvious and hence some questions arise including how the distribution of the parallel applications over the cluster affects the coscheduler performance, how the inter-arrival time affects the turnaround time of the parallel applications, and finally, whether it is worth applying a complex scheduling policy and, if so, which. Our main goal in this work is to shed some light on those questions.

3 CISNE: Cooperative and Integrated Scheduler for Non-dedicated Environments

In order to provide a system that merges space and time sharing scheduling, we propose a new integral system called CISNE. The time sharing scheduling is done by a dynamic coscheduling technique, named CCS (Cooperating CoScheduling) [5], developed previously by our group. Concerning about the space scheduling problem, we present a system called LoRaS. This system is responsible for distributing parallel applications throughout the cluster using information about the system state, the applications to be launched and the CCS characteristics.

Fig. 1 shows the integration of CCS and LoRaS into CISNE. It shows the main components making up the virtual machine. As we can see in the figure, the interaction between the nodes follows a master-slave paradigm. There is one server node (master with the most important control and management functions), and the remaining ones interact with the server in a client (slave) mode.

In the following sections, the CCS and LoRaS systems are explained separately.

3.1 CCS (Cooperating CoScheduling) System

Our T.S. system provides an execution environment where the parallel applications could be dynamically coscheduled. Besides, the given resources are balanced and the interactive responsiveness of the local applications is totally preserved. In order to reach this situation, CCS uses the architecture shown in fig. 1, where each module goal is:

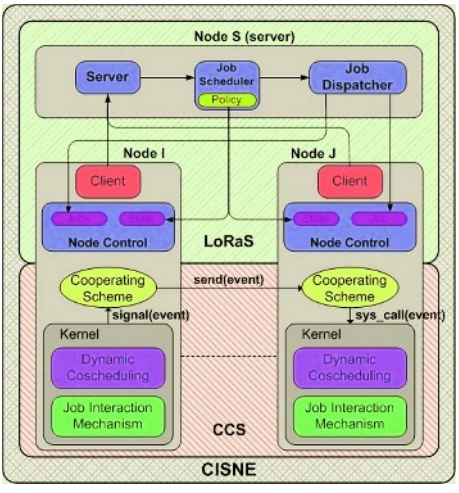

Fig. 1. LoRaS-CCS Architecture

- *Dynamic Coscheduling*: no processes should wait for a non-scheduled process for synchronization/communication. This is achieved by means of increasing the communicating task priority, even causing CPU preemption. (implemented inside the Linux kernel).
- *Job Interaction Mechanism* (JIM): preserves the local user tasks responsiveness. In order to reach its goal, this module manages the amount of resources (CPU and memory) given to the parallel tasks in the node. This is done by means of a *social contract* [9], which establish the amount of resources that could be given to the parallel and local loads, when the node is not idle (implemented inside the Linux kernel).
- *Cooperating Scheme*: this module collaborates with the JIM module in order to balance the resources (memory & CPU) given to the parallel applications throughout the cluster. It is responsible for the exchange of several events, such as the login or logout of a local user into a specific node, or the stopping (restarting) event generated by the JIM module for a specific parallel application. This happen whenever it has to preserve the local responsiveness (implemented in user space).

3.2 LoRaS (Long Range Scheduler) System

LoRaS implements a Job Scheduler in the user space, which provides a Space-Sharing scheduling mechanism. The following is the description of the LoRaS modules shown in fig.1:

- *Client*: sends a job execution request (JER) to the server module on behalf of a parallel user.
- *Server*: the admittance of new JERs to be executed in the system is performed by the *server* module. This JER is then forwarded to the *Job Scheduler* module.

- *Job Scheduler*: executes every received JER using the configured policy. It is important to mention that JER execution is conditioned by the cluster state. If there is no possibility of executing the job on its arrival, then the petition has to wait in a queue for the requested resources.
- *Policy (submodule)*: establishes the possibility of executing a JER for a given cluster state and the JER resources request. This module is designed in such a way that it is easy to change its functionality and hence the LoRaS scheduling system.
- *Job Dispatcher*: considering that every job can have its own characteristics (e.g. a PVM or MPI job), it is necessary to configure the job before launching it. Hence, this module is responsible for doing these previously required tasks.
- *Node Control*: this module has two different functions. On one hand it launches and controls the job execution. On the other hand, it gathers information from the node state and informs the *Job scheduler* (and hence, the *policy* submodule) so that it can take better scheduling decisions.

3.3 Implemented and Evaluated Policies

In this section, we propose several S.S. techniques oriented towards non-dedicated clusters. Unlike traditional techniques oriented to dedicated cluster, all our proposals are characterized by the fact of taking the cluster state into account.

The first proposed policy, named *Uniform*, is characterized by the following: (a) it merges differently oriented applications (i.e. communication or computation) in the same node and (b) it runs applications one over another in an ordered manner, whenever possible. By doing this, we expect to increase the coscheduling probability of the CCS system. By *ordering the applications* we mean to launch parallel applications in such a way that each task of a couple of parallel applications runs in the same set of nodes. This situation is depicted in fig. 2.a and we call it a *Uniform* situation.

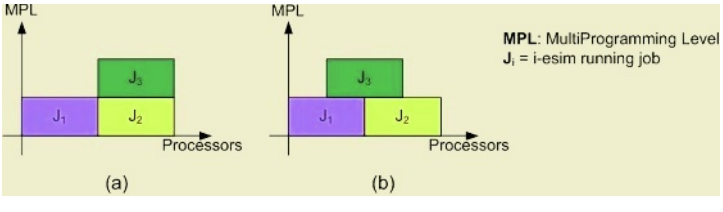

Fig. 2. Difference between a *uniform* (a) and a *normal* (b) policy

However, the *Uniform* policy executes the applications in any free place if there is no space for them in a *uniform* place. Besides, it is important to mention that in every case the policy must try to help to preserve the local user activity by not overloading nodes with some local tasks. This is done by limiting the amount of usable memory and the MPL, respecting the established social contract.

The problem of arranging different size (number of needed nodes) applications in a uniform way, was dealt with by always arranging smaller applications over bigger ones. Therefore, little applications can start sooner, while bigger applications do not notice to much effect from the coscheduling point of view.

The second proposed policy, termed *Normal*, considers the state of the system nodes, but does not consider the running job distributions as the uniform does. Thus, the resulting scheduling can reach a situation like the one depicted in fig. 2.b. where an application like J_3 shares its nodes with a couple of applications. This situation tends to diminish the coscheduling system performance and hence the application execution time is increased.

In addition, both *normal* and *uniform* policies are compared against a *Basic* policy where we execute the parallel workload with an MPL = 1, which means at most one parallel task per node. Finally, and in order to compare with a well known S.S. policy, we introduce an *EASY backfilling* [10] policy in our evaluation. The *EASY* policy executes a job not-at-the-head of the jobs queue, whenever this does not delay the start of the job at the head. By including this policy, we can show the effect of incrementing the MPL compared with the use of an *EASY* policy with an MPL = 1.

It is important to note that for every evaluated policy, we use a FCFS policy for queuing each arriving job. Doing this, we ensure the absence of starvation in the system and a fair treatment for every job.

4 Experimentation

This experimentation is divided into two sections. The first section compares our coscheduling system in relation to traditional coscheduling systems based exclusively on communication events. The second set of results shows how CISNE performs under our defined S.S. policies.

In order to simulate a non-dedicated cluster, we need two different kinds of workloads. On one hand, we need to simulate local user activity and, on the other hand, we need some parallel applications that arrive at some interval.

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. In order to assign these values in a realistic way, 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 needs on inter-activeness (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 alternate CPU activity with interactivity by means of 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 values observed in the monitoring, we loaded the 25% of the nodes with local workload in our experiments.

The parallel workload was a list of 90 NAS parallel applications with a size of 2, 4 or 8 tasks that reached the system following a Poisson distribution [2]. The chosen NAS applications were: CG (mem: 55-120MB / CPU: 65-70% / time: 37-51 sec.), IS (mem: 70-260MB / CPU: 58-69% / time: 40-205 sec.), MG (mem: 60-220MB / CPU: 82-89% / time: 26-240 sec.) and BT (mem: 7-60MB / CPU: 85-93% / time: 90-180 sec.). The parallel jobs were merged so that the entire workload had a balanced requirement of computation and communication (25% of the workload composed by each application). It is important to note that the MPL reached for the workload depended on the system state at each moment, but in no case it surpassed an MPL = 4. This was established in order to respect the social contract, which was set to 50% of the resources available for each kind of load (local/parallel) [5].

Both workloads were executed in an Linux cluster composed of 16 P-IV (1,8GHz) nodes with 512MB of memory and a fast ethernet interconnection network.

4.1 Evaluating the Time-Sharing Systems

In this section we have compared the CCS policy in relation to the plain Linux scheduler and two well known communication-driven coscheduling strategies: implicit and (isolated) dynamic coscheduling. In implicit coscheduling, a process waiting for messages spins for a determined time before blocking. In contrast, dynamic coscheduling deals with all messages arrivals (like CCS, but without resource balancing). It works by increasing the receiving task priority, even causing CPU preemption of the task being executed inside.

They were evaluated by running the parallel workload for several values of MPL (1 to 4). The parallel workload was executed applying a *Normal* S.S. policy. Its performance was measured by means of the slowdown. This is the response-time ratio of a job in a non-dedicated system in relation to the time needed in a system dedicated to this job.

From fig. 3.a, we can see that the slowdown of the parallel applications is always better for our CCS coscheduling system. In fact, this difference increases with the value of the MPL. This good CCS behavior is due to the interaction of the coscheduling scheme with the adaptive and balanced resource allocation carried out by CCS. In addition, the social contract implemented by CCS maintains the response time (measured by the mean of the local benchmark system call latency in fig. 3.b) always under 400ms. This limit for the Response Time, established by [11], is an acceptable threshold before the user can notice a lack of inter-activeness.

These results encouraged us to use CCS to integrate a coscheduler into the CISNE system.

4.2 Evaluating the CISNE Integrated System

In this subsection, we want to show the performance of CISNE, by applying the described space-sharing policies to the CCS system. This interaction will be

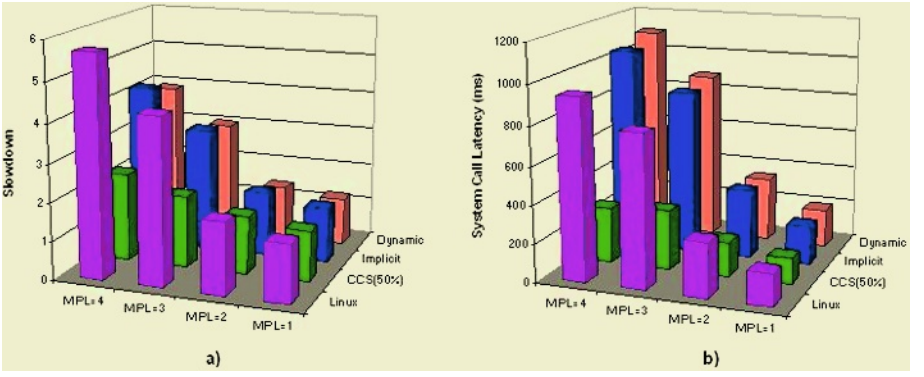

Fig. 3. Parallel applications slowdown (a) and system call latency (b) under the evaluated policies

quantified by measuring the turnaround time of the parallel applications comparing the *Uniform*, *Normal, Basic* and *EASY* policies. In addition, we measure the makespan of the workloads (i.e. the executing time of the whole workload). Doing this it is possible to evaluate CISNE from a system administrator's point of view.

Fig. 4.a shows the turnaround, wait and execution time for every evaluated policy. Here we can see that the *normal* and *backfilling* policies give us almost the same behavior, while the *uniform* policy performs better by reducing the execution time and hence the waiting time of the workload. From this figure, it is also clear that the turnaround time is dictated by the waiting time. On the other hand, it would be desirable to evaluate the effect of the execution time as the predominant turnaround factor. With this aim, we executed the parallel workload doubling the inter-arrival time between applications. Fig. 4.b shows the results obtained for the same policies.

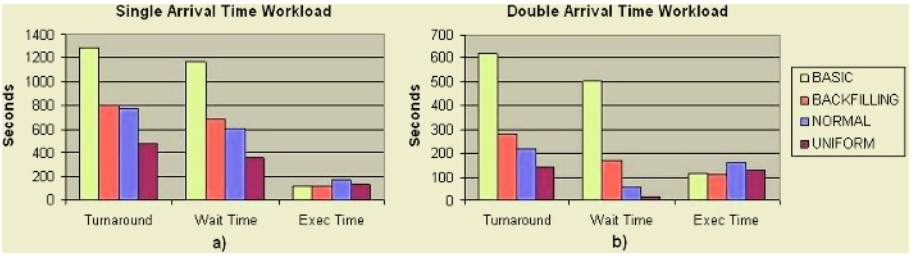

Fig. 4. Turnaround, Wait and Execution time for the exercised workloads

From those figures, it is clear that the job distribution policy has a great impact on the underlying coscheduling system performance, considering the reduction in the execution time. This effect arises for two different reasons: on

one hand, the applications compete for different kinds of resource, letting them evolve without disturbing each other. On the other hand, the fact of merging applications with different communication patterns (under the *Uniform* policy), improves the performance of the coscheduler. This is due to a CCS enhancement in recognizing the communication needs. It is important to note that the execution time for the backfilling and basic policies are better due to the MPL restriction.

Another effect that it is important to mention is how the waiting time is noticeably reduced when we apply a *Uniform* policy. This effect is not only due to a decrease in the execution time, but due to a better resource distribution that enhances the scheduling opportunities. Actually, this effect is not just a benefit of the *Uniform* policy, but a problem of the *Normal* one. The main problem is that the *Normal* policy tends to distribute the resources in such a way that the total available memory throughout the cluster could be enough to execute an application, but there are not enough nodes with enough free memory for launching it. However the *Uniform* policy tends to localize the available resources and then the scheduling possibility is enhanced in the average case. This is due to the elevated percentage of small applications in the workloads tested. That fact was verified in [2] to be representative of the reality.

In order to take a closer look at the enhancement of the coscheduling performance, fig. 5.a shows how the selected policy affects the jobs slowdown. This graphic is calculated by comparing the *Normal* and *Uniform* policies with the *Basic* policy (*Slowdown* = 1), where every job is executed in isolation (except for some local activity). The figure shows how a uniform policy could reduce the slowdown from 40% (1,40) to less than 15% (1,15). This demonstrates the good performance of our coscheduling system as the close interaction with the S.S technique and, once again, that the level of resources is enough to increase the MPL with almost no detriment to the (parallel) application execution time.

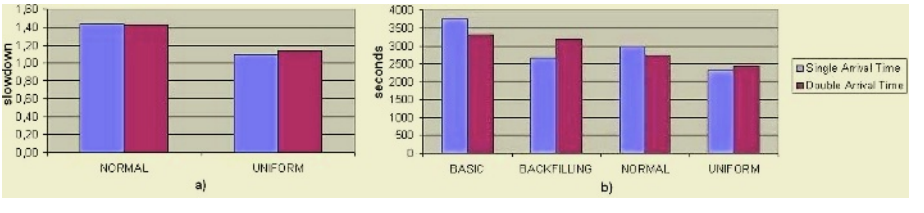

Fig. 5. (a) Applications slowdown for the *Normal* and *Uniform* policies compared with the *Basic* policy. (b) Workloads Makespan for the evaluated workloads and policies

Another aspect we want to analyze is the CISNE behavior from the system point of view (makespan). The results for both workloads (i.e. single and double arrival time), can be observed in fig. 5.b for the policies evaluated.

A couple of effects can be extracted from the figure. First of all, a *backfilling* policy behaves better with a shorter workload arrival time than with a longer

one. This is due to a longer (waiting) jobs queue that enhances the backfilling opportunities. Considering the Normal and Uniform policies, it is clear that the last one has some advantages. In this case, the effect is directly related to a better resource usage and the enhancement in the application turnaround time.

5 Conclusions and Future Work

This work presents a new integral system, named CISNE, that considers both S.S and T.S. concerns, which is applied on a non-dedicated cluster. Using this framework, the paper analyzes how the performance of a dynamic coscheduling system could be affected by the distribution policy over a non-dedicated cluster. With this aim, we evaluated four policies oriented to non-dedicated clusters: *Uniform*, *Normal*, *Backfilling* and *Basic*. We found that a *Uniform* policy (i.e. a set of applications running on the same set of nodes), can dramatically diminish the turnaround time of the applications (up to 76%) compared with other approaches. In addition, the performance of a uniform distribution was evaluated considering a turnaround time limited, on one hand, by the waiting time (single arrival time workload), and on the other hand by the execution time (double arrival time workload). In both scenarios a *Uniform* policy was shown to perform well, even from the system point of view (makespan). It is important to note that those gains were obtained without disturbing the system responsiveness.

Considering our future work and taking into account that the *Uniform* and *EASY* policies attack the scheduling problem from different points of view, they could be combined in a schema where the MPL is greater than one and we also apply a backfilling policy. To do this, we have to define a prediction model to establish the execution time of a parallel application considering the cluster state and the interaction between the running applications.

References

1. Y. Zhang, H. Franke, J. Moreira and A. Sivasubramaniam. “An Integrated Approach to Parallel Scheduling Using Gang-Scheduling, Backfilling and Migration”. *IEEE Transactions on Parallel and Distributed Systems, 14(3):236-247, March 2003.*
2. D. G. Feitelson. Packing schemes for gang scheduling. *In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, 1996, Lect. Notes Comput. Sci. vol. 1162, pp. 89-110.*
3. T. Anderson, D. Culler, D. Patterson and the NOW team. “A case for NOW (Network of Workstations)”. *IEEE Micro, Vol. 15, pp. 54-64. 1995.*
4. C. Anglano. “A Comparative Evaluation of Implicit Coscheduling Strategies for Networks of Workstations”. *In 9th International Symposium on High Performance Distributed Computing (HPDC’2000), pp.221-228, 2000.*
5. M. Hanzich, F. Giné, P. Hernández, F. Solsona and E. Luque. "Coscheduling and Multiprogramming Level in a Non-dedicated Cluster". *EuroPVM’2004, LNCS, vol. 3241, pp. 327-336, 2004.*

6. F. Solsona, F. Giné, P. Hernández, E. Luque. "Implementing Explicit and Implicit Coscheduling in a PVM Environment". *EuroPar 2000, LNCS, Vol 1900, pp 1164-1170. 2000.*
7. Ousterhout, J. . "Scheduling techniques for concurrent systems". *Proceedings of the Conference on Distributed Computing Systems. 1982.*
8. G. Choi, S. Agarwal, J. Kim, A. Yoo and C. Das. "Impact of Allocation Strategies on Communication-Driven Coscheduling in Clusters". *EuroPar 2003: 160*-168.
9. R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson and D.A. Patterson. "The Interaction of Parallel and Sequential Workloads on a Network of Workstations". *ACM SIGMETRICS'95, pp.267-277, 1995.*
10. A. W. Mu'alem and D. G. Feitelson. "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling ". *IEEE Trans. Parallel & Distributed Syst. 12(6), pp. 529-543, Jun 2001.*
11. Nielsen. "Advances in Human-Computer Interaction". J. Nielsen, J. (ed.), Intellect Publishers, 1995.