Job Scheduling considering Best-Effort and Soft Real-Time Applications on Non-dedicated Clusters

J.García, M.Hanzich, P. Hernández, E. Luque*
Dept. de Informática, Universitat Autònoma de Barcelona
Bellaterra, 08290, Spain
{jrgarcia,mauricio}@caos.uab.es, {emilio.luque,porfidio.hernandez}@uab.es

and

J. Lérida, F.Giné, F.Solsona
Dept. de Informática e Ing. Industrial, Universitat de Lleida
Lleida, 25001, Spain
{jlerida,sisco,francesc}@diei.udl.es

Abstract

As Network Of Workstations (NOWs) emerge as a viable platform for a wide range of workloads, new scheduling approaches are needed to allocate the collection of resources from competing applications. New workload types introduce high uncertainty into the predictability of the system, hindering the applicability of the job scheduling strategies. A new kind of parallel applications has appeared in business or scientific domains, namely Soft Real-Time (SRT). They, together with new SRT desktop applications, turn prediction into a more difficult goal by adding inherent complexity to estimation procedures. In previous work, we introduced an estimation engine into our job scheduling system, termed CISNE. In this work, the estimation engine is extended, by adding two new kernels, both SRT aware. Experimental results confirm the better performance of simulated respect to the analytical kernels and show a maximum average prediction error deviation of 20%.

Keywords: parallel processing, soft real-time, job scheduling, non-dedicated clusters.

Resumen

Mientras las Redes de Estaciones de Trabajo (NOWs) emergen como una plataforma viable para un amplio espectro de aplicaciones, son necesarios nuevos enfoques para planificar los recursos disponibles entre las aplicaciones que compiten por ellos. Los nuevos tipos de cargas introducen una alta incertidumbre en la predictibilidad del sistema, afectando la aplicabilidad de las estrategias de planificación de tareas. Un nuevo tipo de aplicaciones paralelas, denominado tiempo real débil (SRT), ha aparecido tanto en los ámbitos comerciales como científicos. Las nuevas aplicaciones paralelas SRT, conjuntamente con los nuevos tipos de aplicaciones SRT de escritorio, convierten la predicción en una meta aún más difícil, al agregar complejidad a los procedimientos de estimación. En trabajos anteriores dotamos al sistema CISNE de un motor de estimación. En este trabajo añadimos al sistema de predicción fuera de línea dos nuevos núcleos de estimación con capacidad SRT. Los resultados experimentales muestran un mejor rendimiento del núcleo simulado con respecto a su homólogo analítico, mostrando un promedio de desviación máximo del 20%.

Keywords: procesamiento paralelo, tiempo real débil, planificación de tareas, clusters no dedicados.

*This work was supported by the MEyC-Spain under contract TIN 2004-03388.

1 INTRODUCTION

Nowadays, wasted computational resources are a common reality in open laboratories using NOWs in any university. The aim is to take advantage of those available resources to do parallel computation [1]. The possibility of using this computing power to execute distributed applications with a performance equivalent to a Massively Parallel Processor (MPP) and without perturbing the local user's applications performance has led to proposals for new resource management environments [7]. As a try to take advantage of these idle computational resources, these environments combine space sharing and time sharing scheduling techniques. In order to provide such a system, we developed CISNE [7].

CISNE is composed basically of a dynamic coscheduling technique and a space sharing scheduler. The coscheduling technique [5] ensures the progress of running parallel jobs without disturbing the local users, even when using a Multiprogramming Parallel Level (MPL) greater than one [5]. The space sharing scheduler is named LoRaS and is component of CISNE that performs the parallel workload distribution among the cluster nodes. LoRaS takes into account the state of the cluster and the characteristics of the local and parallel workload to implement different policies used in job allocation. To achieve better scheduling decisions, CISNE needs to foresee the state of the cluster. Likewise, this prediction capacity could help to guarantee some limits in the turnaround and missdeadline time to the applications of any user.

Unfortunately, new desktop SRT applications [4] turn prediction into a more difficult goal, introducing great uncertainty into the predictability of the system. A situation that gets worse with the new kind of parallel SRT applications that has arisen in business and scientific domains [13]. This new kind of parallel applications usually requires bounded response time or guaranteed turnaround time [9], [12]. In this context, it is mandatory to ensure that Best-effort applications do not affect the deadline requirements for the SRT jobs, and to protect the Best-effort applications from starvation.

According to this new situation, we are interested in researching new estimation methods, which take Best-effort and SRT applications, local and parallel, into account. Following studies like [8], [11], the choice was a simulation approach to represent our scheduling system. These systems have an estimation kernel, which is in charge of the prediction of the execution time. Unlike previous works, where these kernels are based on the use of a historical repository, we bet by the use of estimation kernels based on simulation or analytical methods. Both proposals were evaluated experimentally. Our results reveal as the simulated kernel achieves better results due to its high degree of detail.

The outline of this work is as follows, section 2 introduces the SRT application framework used in this study, also places our work in the job scheduling area. Section 3 depicts the off-line simulation process in the CISNE, and section 4 describes the new SRT aware approaches for calculating the Remaining Execution Time (**RExT**). Next, section 5 analyzes the experimental results. Finally, the conclusions and the future work are explained.

2 BACKGROUND

This section introduces the novel application framework studied in this paper. Likewise, this also locates our work in the job scheduling area.

2.1 Local and Parallel SRT Applications

An extensive work has been done in Real Time (RT) support [10], mainly directed towards industrial environment. Recently, this knowledge has been gradually migrated to the commercial world. Most of researches efforts focuses in the idea that common operating systems performs badly in presence of applications requiring some kind of resource's priority, usually multimedia applications, such as videos [3], [14], [15].

A common definition of a SRT application is a RT application that may loose some deadlines and no disaster will occur. In the local user environment, a multimedia application suits this definition quite well. It means that nothing happens if a video skips a few frames. The same idea fits for parallel SRT applications. Some users want its results within a deadline but it is not always feasible, maybe a system overload would interfere and cause a missdeadline. However, these undesirable situations should be minimized. According to this aim, neither local nor parallel SRT applications can be ignored. Thus, we will try to fulfill their requirements in the best manner.

Local SRT applications may be *periodic* or *aperiodic* [10]. The periodic tasks are those that are released regularly in periods and must be executed before a certain deadline smaller than the period. Aperiodic tasks are those that are activated irregularly at some unknown and maybe unbounded rate.

From now on, parallel SRT applications will be denoted as par_SRT and local SRT applications as local_SRT.

2.2 Job Scheduling and SRT Applications

In [13], new types of par_SRT are considered and their behavior studied for dedicated clusters by using simulation. Given that the cluster is dedicated, any kind of local load is considered. Regarding to the parallel SRT applications, RT vision applications are taken into account.

Studies like [9], [12] are focused in new types of parallel SRT application, both working in dedicated *Beowulf* clusters. The type of par_SRT considered are RT constrained vision applications in [12] and sensors collecting large amounts of data in [9].

Our work is focused in a novel research area, considering non-dedicated clusters. It means that parallel applications, which can be either par_SRT or not, together with local workload, which can also be either Best-effort or SRT, are running all together in the same cluster, sharing the same computational resources.

3 ENVIRONMENT CONSIDERATIONS

In order to implement and evaluate our estimation proposals, we need a scheduling system oriented towards non-dedicated environments. With this goal, in previous works we developed the CISNE system [7] as an integral scheduling environment that merges both Time and Space sharing subsystems. This section describes this system to achieve a better understanding of the off-line simulation process.

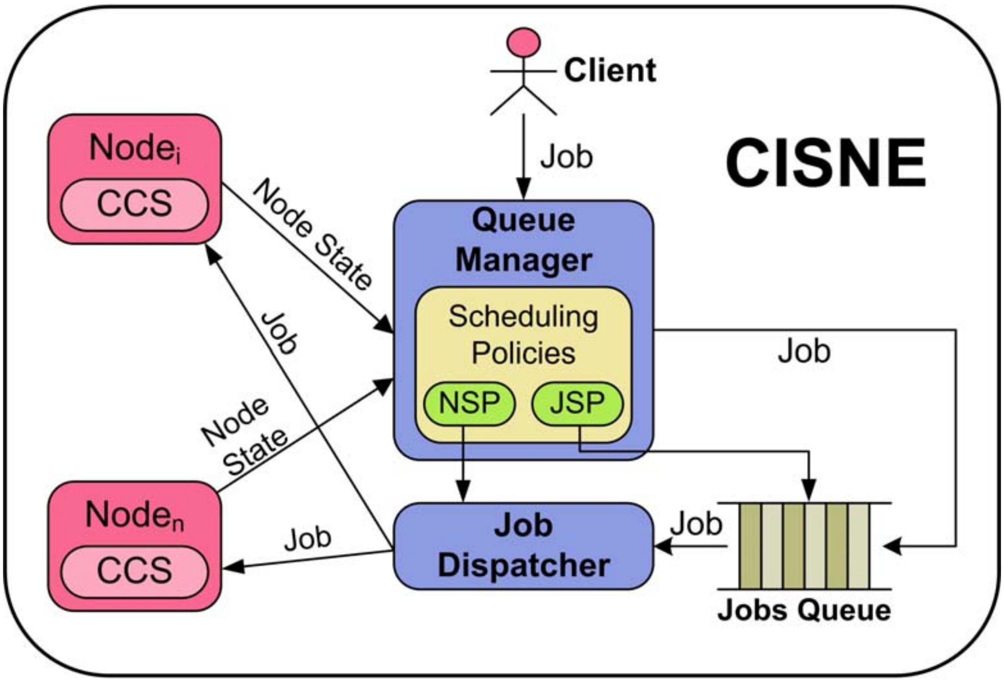

Figure 1. General architecture of the CISNE system.

3.1 Description

The CISNE architecture is depicted in Figure 1. Its main aim is to manage parallel applications in a non-dedicated environment, ensuring benefits for the parallel applications, while preserving the local task responsiveness. When a parallel job is submitted to the CISNE system, the job waits in a queue until the *Queues Manager* decides to schedule it. This decision is taken according to the computational requirements of each parallel job waiting in the queue, together with each node state information received. According to the above description, the CISNE system needs to define the following policies:

- *Job Selection Policy* (JSP) is the policy for selecting the next job to run from the jobs queue. This could depend on the order in the queue (i.e. First-Come-First-Served, *FCFS*), and the estimated cluster state (intrusion level into the local workload, the MultiProgramming Level (MPL) of parallel applications, the memory and CPU usage on each node; i.e. Backfilling). In order to make easy the comprehension of the experimental results, a simple FCFS policy is used through this work.

- *Node Selection Policy* (NSP) is the policy for distributing the parallel tasks among the nodes. This depends on the cluster state and the parallel job characteristics. In this work, we use a policy, named *Normal*, which selects the nodes for executing a parallel application considering only the resources usage level throughout the cluster. This policy does not overload any node in detriment of the local user interactiveness. To achieve this, it establishes an acceptable system usage limit for CPU and main memory by means of a social contract [2] between local and parallel users.

In order to evaluate different workloads and scheduling policies over several simulated cluster environments, an off-line simulation system was included in the CISNE system. Next subsection describes the off-line simulation engine of CISNE.

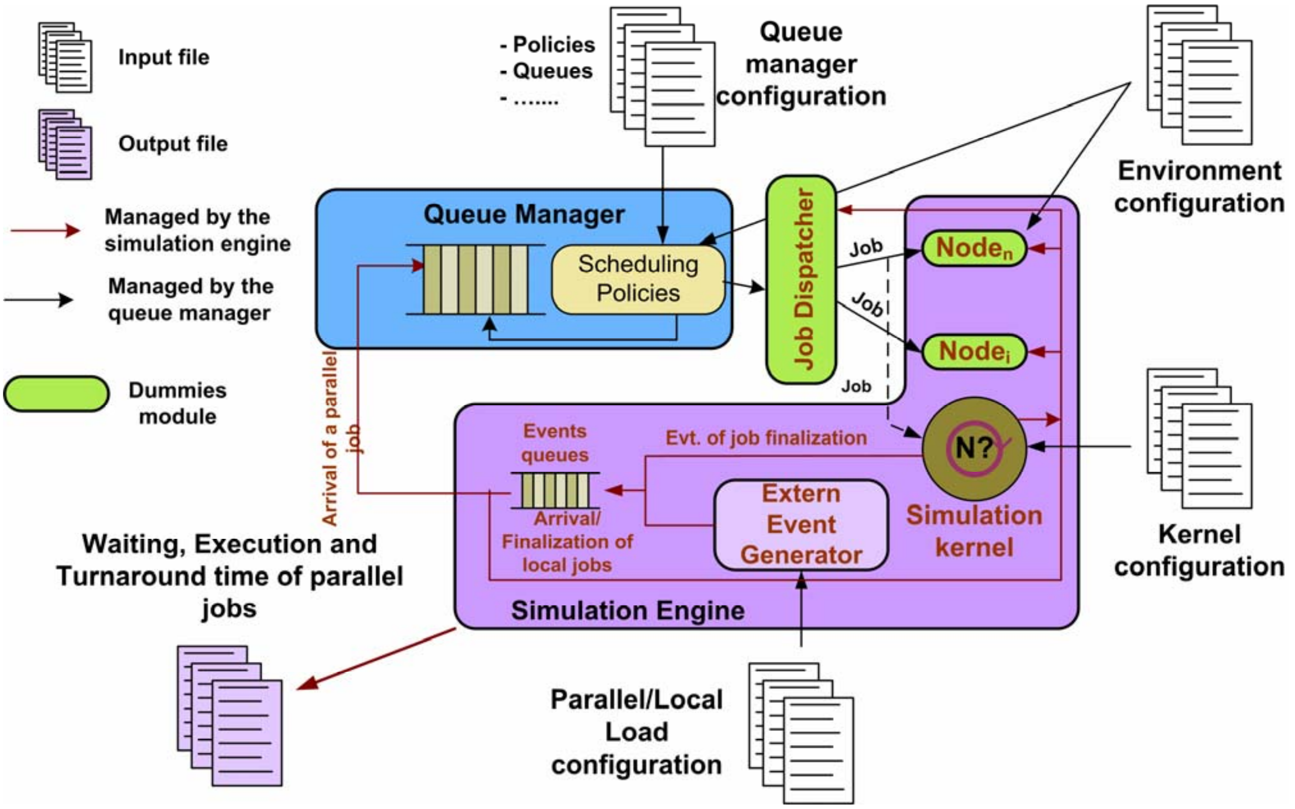

Figure 2. Architecture of the Off-line simulator.

3.2 Off-line Simulation

According to Figure 2, the architecture of the simulator is made up of two main modules. The *Queue Manager*, who manages the Waiting Queue of the system, which means selecting the next job to run and distributing its tasks among the nodes in the cluster. The other main module is the *Simulation Engine*. As shown in Figure 2, our simulator is directed by discrete events, such as the arrival, scheduling or finalization of any kind of jobs. In order to be able to simulate, the Simulation Engine has three different input configuration files:

- *Environment Configuration*: specifies the set of nodes used by the Queue Manager to execute the parallel workload. It describes the number of nodes in the simulated environment and their main characteristics: computational power, memory size and the local load resource usage.
- *Kernel Configuration*: allows the method used to estimate the turnaround time of each parallel job managed by the Queue Manager to be set. It can be either the analytical or simulated estimation method, described in next section.
- *Load Configuration*: allows specification of the list of parallel and local jobs to be simulated. By each job, our simulator needs detailed information, for example: job size, execution time in isolation, arrival time and its maximum requirements of CPU and memory.

Once all the configuration files are loaded, CISNE is ready to start the simulation process, using the application information collected before from real executions. This information, together with the cluster state, is used to generate data sets that conforms the different stages of simulation. A new stage begins with each arrival of a new parallel application. In such a moment, a data set containing current cluster state is created and the new parallel application is included in it.

These kernels, represented by *Simulation kernel* in Figure 2, must return an estimation of the Remaining Execution Time (**RExT**) for the running applications. From the RExT value, our simulator is able to obtain the Turnaround time of each parallel application. It is worth pointing out that the off-line simulation method has been validated in [6],[7].

4 SRT CAPABLE KERNELS ADDED

In order to provide SRT capabilities to the off-line simulation of CISNE, two new estimation kernels were developed, one analytical and the other simulated. These new kernels are described in this section.

4.1 Analytical Method

To make easier to understand our SRT aware analytical kernel, termed $RExT_{ANL-SRT}(j)$, we will describe a non-SRT capable analytical kernel used as seed.

The non-SRT capable method, termed $RExT_{CPU}(j)$ starts by calculating the RExT that the application would need if it were executed in isolation ($RExT_{isol}(j)$). This value is calculated as follows:

$$RExT_{isol}(j) = \frac{t_{tot}(j) \times (t_{Tcpu}(j) - t_{Ucpu}(j))}{t_{Ucpu}(j)}, \tag{1}$$

where $t_{tot}(j)$ is the total execution time, $t_{Tcpu}(j)$ is the amount of CPU time by parallel application *j* running in isolation and $t_{Ucpu}(j)$ is the amount of CPU time used by parallel application *j* from its beginning, a value provided by the *Queue Manager* module.

Note that equation 1 assumes that the $RExT_{isol}(j)$ is proportional to the total execution time in isolation ($t_{tot}(j)$) bounded by the CPU time to be consumed ($t_{Tcpu}(j) - t_{Ucpu}(j)$), and the total amount of CPU time needed by the application ($t_{Tcpu}(j)$).

Next step in $RExT_{CPU}(j)$ method is to consider the CPU requirements of the tasks (in percentage). According to this, the $RExT(j)$ is calculated as follows:

$$RExT_{CPU}(j) = RExT_{isol}(j) \times \frac{CPU(j)}{CPU_{feasible}(j)}, \tag{2}$$

where $CPU(j)$ is CPU percentage ($t_{Tcpu}(j)/t_{tot}(j)$) that the application can use and

$$CPU_{feasible}(j) = \min(CPU(j), \frac{CPU(j)}{CPU_{\max}(j)}) \tag{3}$$

is the maximum CPU usage (in percentage) that we expect the application *j* could use. Finally

$$CPU_{\max}(n) = \max(CPU_{par}(n) + CPU_{loc}(n) \mid n \in N(j)) \tag{4}$$

where $CPU_{loc/par}(n)$ is the sum of the CPU usage of each local/parallel task running in the node *n*. Note that this represents the maximum CPU usage requirements (in percentage) among the nodes where the application *j* is running.

Starting from the $RExT_{CPU}(j)$, we describe our SRT aware analytical proposal, which considers not only the CPU usage, but distinguishes which of the requirements of the parallel and local tasks are SRT. In order to do this, we redefine the $CPU_{feasible}(j)$ expression of $RExT_{CPU}(j)$ method as follows:

$$CPU_{feasibleSRT}(j) = \begin{cases} CPU(j) & j \in S_{srt} \\ \min(CPU(j), CPU_{mS}(j)) & j \notin S_{srt} \end{cases} \tag{5}$$

where S_{srt} is the set of parallel SRT applications currently running in the cluster and $CPU_{mS}(j)$ is defined as

$$CPU_{mS}(j) = \min(\frac{CPU(j) \times (100 - CPU_{SRT}(n))}{CPU_{nonSRT}(n)} \mid n \in N(j)) \tag{6}$$

where $CPU_{SRT}(n)$ and $CPU_{nonSRT}(n)$ are the sum of CPU required by every SRT and non-SRT tasks, respectively, running in the node *n*. The tasks may be either parallel or local.

4.2 The Simulated Approach

Our second SRT aware proposal to compute RExT, which is named $RExT_{SIM-SRT}(j)$, is based on simulation. An external simulator, named *Simulation Kernel* in Figure 2, does the simulation of the successive stages generated by the CISNE estimation system.

Like the analytical estimation method, it takes a snapshot of a simulation point but performs a detailed simulation instead of analytical calculations. $RExT_{SIM-SRT}(j)$ method may assign SRT tasks priorities using several policies, Rate Monotonic Scheduling (RMS [10]) is the policy used in this study. Non-SRT tasks priorities (parallel or local) are assigned using Round Robin.

$RExT_{SIM-SRT}(j)$ method is in fact a whole simulation engine, capable of performing a low level simulation of each node behavior, taking into account available resources and allocation policies. The communication between the two simulation engines, the off-line simulator and the simulation kernel (see Figure 2), is achieved through XML format files containing cluster state and simulation results. In each stage, the simulation kernel takes the cluster state as input and returns the RExT of each parallel job to the off-line simulator.

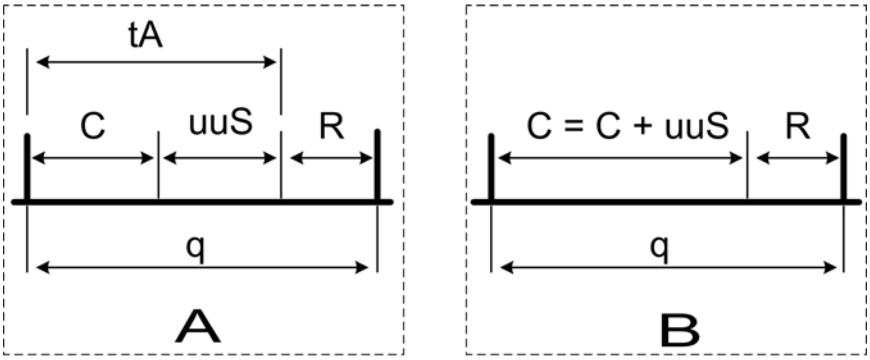

Figure 3. Dynamic slice in per-stage simulator.

Due to space restrictions, only the way $RExT_{SIM-SRT}(j)$ method deals with local_SRT and par_SRT jobs is described more deeply. To simplify our model and be able to predict in our system, we convert par_SRT deadline (*D(j)*) into a period ($T_{par}(j,n)$) and a computation time ($C_{par}(j,n)$). The period is set as a default value, and the computation time is calculated using equation 7.

$$CPU_{mS}(j) = \min\left(\frac{CPU(j)\times(100 - CPU_{SRT}(n))}{CPU_{nonSRT}(n)} \mid n \in N(j)\right) \quad (7)$$

where $t_{exec}(j)$ is the current running time for par_SRT *j*. This equation represents a way to ensure that the par_SRT will finish its execution at least within the desired deadline, achieved by reserving to par_SRT the minimal quantum slice it needs.

This is a pessimistic approach and causes the par_SRT to finish very close to its deadline, no matter the cluster load. In order to palliate this effect, the computation time assigned to par_SRT of each CPU quantum is assigned dynamically as the bigger value of the maximum quantum slice available to the par_SRT and the quantum slice needed by the par_SRT according to equation 7.

The dynamic slice process is depicted in Figure 3, where **q** is the CPU quantum, **C** is the reserved computation time to a par_SRT, **R** is the reserved computation slice of any other SRT applications, local or parallel, **uuS** is the unused slice of the quantum and the **tA** represents the available slice to par_SRT. Whenever is possible our simulator assigns all available quantum slice to par_SRT (represented by the **tA** in Figure 3.A), taking into account the CPU needs of other SRT jobs present in the node, local or parallel. Figure 3.B shows a dynamically assigned computation time, which is the sum of **uuS** and **C**, i.e: **tA**. In this new approach, the value calculated using equation 7, **C** in Figure 3, is used as the minimal computation time that the par_SRT must receive. This way a par_SRT may take advantage of low CPU load in a node at any moment.

The comparative values of our two SRT aware estimation kernels described above are shown in the next section.

5 EXPERIMENTATION

In order to carry out the experimentation process, we need two different kinds of workload. On one hand, we need to simulate the local user activity and, on the other, we need some parallel applications arriving at a representative interval.

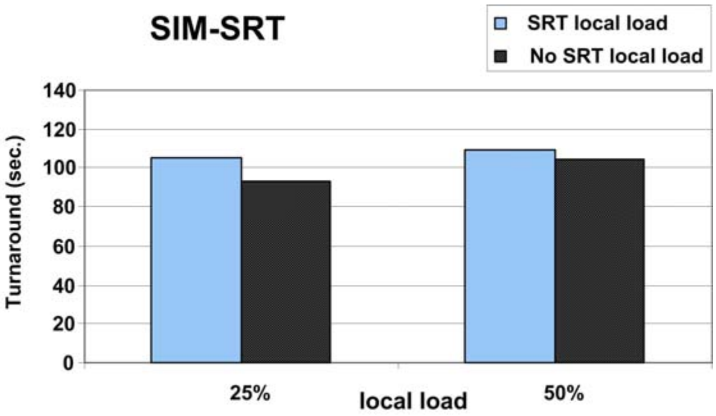

Figure 4. Influence of the local SRT load into the parallel applications performance measured for the $RExT_{SIM-SRT}(j)$ method.

The local user activity is modeled by a benchmark that could be parametrized in such a way that it uses a percentage of CPU, memory and network; which represents the Best-Effort applications. To parametrize this benchmark realistically, we measure our open laboratories for a couple of weeks and use the collected values to run the benchmark (15% CPU, 35% Mem., 0,5KB/sec LAN). We used the CPU (11% or 41%) and memory (15%-20%) requirements of the Xine video player for different visualization window sizes [4] to represent the SRT local workload.

The parallel workload was a list of 90 PVM NAS parallel jobs (CG, IS, MG, BT) previously characterized in our system with a size of up to 16 tasks, which reached the system following a Poisson distribution. These jobs were merged so that the entire workload had a balanced requirement for computation and communication. It is important to mention that the *MPL* reached for the workload depends on the system state at each moment, but in no case will exceed an *MPL*=4 [6]. In this case, and to represent a parallel SRT workload we used the same parameters as for the non-RT parallel applications (execution time in isolation, memory usage, CPU consumption, etc.), but defining a deadline for some of the parallel applications contained in the workload.

5.1 Results

The Figures 4 and 5 shows the influence of the local SRT load on the performance of the parallel applications for both SRT capable estimation methods. All simulation was carried out for 16 nodes. As can be observed there is a difference of near 15% between the best case ($RExT_{SIM-SRT}(j)$ estimation method with 50% of the nodes with some local SRT load, and the worst case ($RExT_{SIM-SRT}(j)$ estimation method with 25% of the nodes with some local SRT load). As expected, turnaround variations depend not only on the estimation method, but also on the local load. A consideration that should be taken into account is the minor turnaround (14% lower in average) returned by the simulation estimation method (Figure 4) compared with the analytical (Figure 5). That effect combined with the fact that $RExT_{SIM-SRT}(j)$ is a pessimistic estimation method, allows us to assume that this prediction is closer to a real execution.

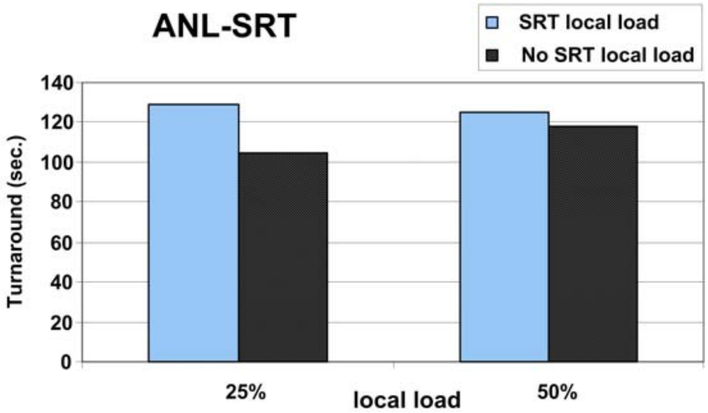

Figure 5. Influence of the local SRT load into the parallel applications performance measured for the $RExT_{ANL-SRT}(j)$ method.

In other experiment we also add parallel SRT applications to the workload for generating Figure 6, where we measure the *Failed Turnaround Percent* of the applications for our two SRT estimation methods. This metric is calculated as the percent of par_SRT that miss theirs turnaround times when considered SRT.

From the Figure 6, it is clear that the simulated method ($RExT_{SIM-SRT}(j)$) performs much better than the analytical one, almost always staying below 20% of missdeadline for the applications.

When local load is considerable (50% a half of which is SRT) and some parallel applications are SRT (30%), the analytical model gives an awfully bad result. The problem relies in the simplicity of the analytical model to estimate a system as complex as a non-dedicated cluster.

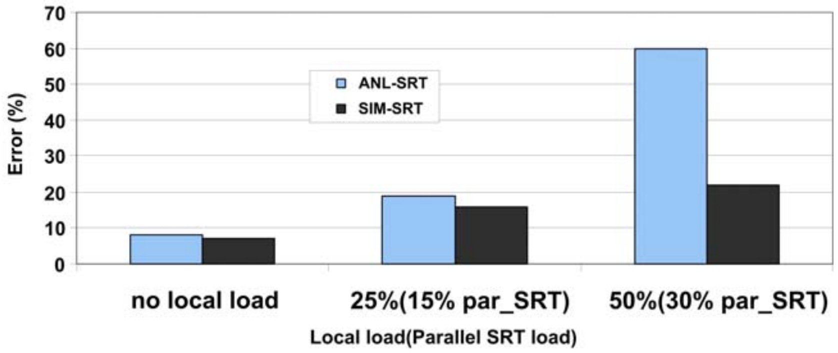

Figure 6. Failed turnaround percent for both SRT estimation methods.

6 CONCLUSIONS AND FUTURE WORK

Due to recent changes in the parallel and local application characteristics, it is urgent to enhance the prediction capacity of a resource management environment over a non-dedicated cluster, to include some sort of SRT estimation schema. In this work we have proposed estimation methods based on analytical and simulation approaches. The main aim of these methods is the prediction of RExT for a given running application and a defined cluster state. Based on such estimation, the system is then capable of estimating the turnaround time of the parallel applications as a parallel user metric.

The results shown in the present work demonstrate that an analytical approach to make some predictions in a non-dedicated SRT environment is possible, as long as local user and par_SRT presence remains low. A simulation approach for making such estimations was shown to be a very good option, even in the presence of high parallel and local SRT loads.

Future work will focus on consider the influence of multi-core processors and a way to take advantage of them in job scheduling. The need to perform exhaustive feasibility analyses of the SRT load is also an open line.

REFERENCES

[1] Acharya, A. and Setia, S. *Availability and Utility of Idle Memory in Workstation Clusters*, Proceedings of the ACM SIGMETRICS, 35-46, 1999.

[2] Arpaci, R. et al. *The Interaction of Parallel and Sequential Workloads on a Network of Workstations*, Proceedings of the ACM SIGMETRICS 1995, 267-277, 1995.

[3] Childs, S. and Ingram, D. *The Linux-SRT Integrated Multimedia Operating System: Bringing QoS to the Desktop*, IEEE Computer Society, 00, 0135, 2001.

[4] Etsion, Y.; Tsafrir, D. and Feitelson, D. G. *Desktop Scheduling: How Can We Know What the User Wants?*, ACM Trans. Multimedia Comput. Commun. Appl., ACM Press, 2, 318-342, 2006.

[5] Gin F. *Cooperating Coscheduling: a coscheduling proposal for non-dedicated, multiprogrammed clusters*, Ph.D. Thesis, Universitat Autónoma de Barcelona, 2004.

[6] Hanzich, M. et al. *Using Simulation, Historical and Hybrid Estimation Systems for Enhancing Job Scheduling on NOWs*, Proceedings of the IEEE International Conference on Cluster Computing, pp 1-12, 2006.

[7] Hanzich, M. et al. *CISNE: A New Integral Approach for Scheduling Parallel Applications on Non-Dedicated Clusters*, *EuroPar 2005*, Lecture Notes in Computer Science, 3648, 220-230, 2005.

[8] Li, H.; Groep, D.; Templon, J. and Wolters, L. *Predicting Job Start Times on Clusters*, Proceedings of CCGrid2004, IEEE Computer Society Press, 2004.

[9] Plale, B.; Turner, G. and Sharma, *A. Real Time Response to Streaming Data on Linux Clusters*, Computer Science Department Technical Report TR-569. Indiana University, 2002.

[10] Sha, L. et al. *Real Time scheduling Theory: A Historical Perspective*, Real-Time Systems, 28, 101-155, 2004.

[11] Smith, W. and Wong, P. *Resource Selection Using Execution and Queue Wait Time Predictions*, NAS Technical Reports, 2002.

[12] Yang, M. et al. *An automatic scheduler for real-time vision applications*, Proceedings of Parallel and Distributed Processing Symposium, 2001.

[13] Zhan, Y. and Sivasubramaniam, A. *Scheduling Best-Effort and Real-Time Pipelined Applications on Time-Shared Clusters*, Proceedings of SPAA'2001, 209-218, 2001.

[14] Hide, E.; Stack, T.; Regehr, J. and Lepreau, J. *Dynamic CPU management for real-time, middleware-based systems*. Proceedings of RTAS 2004. 10th IEEE, 286-295, 2004.

[15] Nieh, J. and Lam, M. *A SMART Scheduler for Multimedia Applications*, ACM Transactions on Computer Systems, 21, 117-163, 2003.

[16] Etsion, Y.; Tsafrir, D. and Feitelson, D. G. *Process prioritization using output production: Scheduling for multimedia*, ACM Trans. Multimedia Comput. Commun. Appl., ACM Press, 2, 318-342, 2006.