A Markovian Sensibility Analysis for Parallel Processing Scheduling on GNU/Linux

Regiane Y. Kawasaki[1], Luiz Affonso Guedes[2], Diego L. Cardoso[1], Carlos R. L. Francês[1], Glaucio H. S. Carvalho[1], Solon V. Carvalho[3], João C. W. A. Costa[1] and Marcelino S. Silva[1]

[1] Department of Electrical and Computing Engineering, Federal University of Pará (UFPA), 66.075-900,
Belém, PA, Brazil
{kawasaki,diego,rfrances,ghsc,marcelino}@ufpa.br,
[2] Department of Computing Engineering and Automation, Federal University of Rio Grande do Norte (UFRN), 59072-970,
Natal, RN, Brazil
affonso@dca.ufrn.br,
[3] National Institute for Space Research (INPE), Computing and Applied Mathematics Laboratory (LAC), P.O. Box 515, 12245-970,
São José dos Campos, SP, Brazil
solon@lac.inpe.br

Abstract. Parallel Computing has become a powerful tool to overcome certain types of computational problems in many areas such as engineering, especially due to the increasing diversity of platforms for execution of this type of application. The use of parallel computing over LANs and WANs is an alternative in the universe of dedicated environments (parallel machines and clusters), but, in some cases, it needs to imply QoS (Quality of Service) parameters, so it can execute efficiently. In this scenario, the deployment of resource allocation scheme plays an important role in order to satisfy the QoS requirements for parallel applications. In this paper we propose and present Markovian models for resource allocation (CPU allocation) schemes in a GPOS (General Purpose Operating Systems), aiming at offering an optimization method which makes the efficient performance of parallel and interactive applications feasible.

1 Introduction

In the last years, the set of platforms for the performance of parallel applications has become diversified. In the beginning, these environments were limited to some processor units linked by internal bus. However, today, typical platforms are great sets of computers linked by many different networks [1]. The points which caused that approach change were [2]: (a) the high cost for achievement and maintenance; (b) the use of highly specific purpose, which usually generates a high degree of idleness, characteristics which limited the achievement of parallel machines.

An alternative to this problem is to consider a differentiated treatment for the parallel processes, whenever demanded, by means of QoS approach [3]. However, provision of QoS guarantees in GPOS or in other similar systems (like wireless and cellular mobile networks) is a complex issue due to problems such as those related to how to design the system behavior which needs process scheduling and admission control. It becomes even more challenging when in a system a specific process or groups of processes need a minimum of resource assurance. To do so, Call Admission Control (CAC) must act together with the process scheduler, so that, when the system detects that a determined application needs a resource assurance, it may adjust the system to provide it [4].

When it is needed to conjecture about the system performance, usually a model s constructed, which obtains its essential information. From this point, a performance analysis is carried out in order to explore and identify the system's behavior, bottleneck, bounds, etc [5]. Due to the necessity of investigating the feasibility of providing QoS to guarantee a minimum of resources to processes that need a differentiated treatment in a general purpose operating system, in case GNU/Linux, a performance model has been developed for the traditional GNU/Linux scheduler. Besides the traditional GNU/Linux model, a different model that reserves a percentage of the processor time for providing attention to parallel tasks has also been developed. By solving these models, their performance evaluation was compared to identify the changes in the system behavior due to reserving the resource.

This paper is organized as follows. In Section 2 it describes the analytical model of Linux scheduler architecture and a new scheduler model with resource allocation. The sensibility analysis is based on a detailed mathematical model followed by numerical results that are presented in Section 3, admitted with or without resource reservation models. Finally, in Section 4 it shows the final remarks of this work.

2 Linux Scheduler Analytical Model

The basic structure of the Linux scheduler is the process queue (*struct runqueue*). This *struct* is defined inside the archive *kernel/sched.c*. The current O(1) scheduler keeps a *runqueue* per processor, which is responsible for containing all the executable processes of a given processor. Thus, if a process is inserted in a *runqueue* of a specific processor, it will only run on that processor [6]. Each *runqueue* contains two priority arrays [7]: active and expired. Priority arrays are data structures composed of a priority bitmap and an array that contains one process queue for each priority.

Linux scheduler and admission control is depicted in Fig.1. Higher (parallel jobs with QoS, kernel process) and lower priority jobs (compilers, browsers, parallel - without QoS jobs, and others) arrive at the system according to two mutually independent Poisson processes with parameters λ_1 and λ_2, respectively. For the sake of simplicity, it is assumed that both services require a negative exponential service time with rate μ. A job is removed from the active array if:

(a) its processing is finished, with rate $qs_1\mu$ (for high priority jobs) and $qs_2\mu$ (for the other processes); or (b) it needs to be rescheduled to the high priority queue or low priority queue in the expired array, with rates $pr_1\mu$ or $pr_3\mu$, respectively. The scheduling of a job in the low priority queue in the active array is tied to the occupancy of the high priority queue in the active array in the sense that it will only be scheduled if the high priority queue in the active array is empty. When the processing queues are empty in an active array and there is a job to be processed in the expired array, these arrays are switched. This switching (via pointer) has an associated time of the $10^{-6}s$ [7].

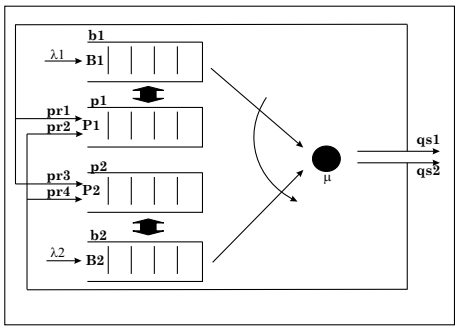


Fig. 1. System model.

Given the assumptions presented above, it a Continuous-Time Markov Chain (CTMC) [8] model of the system, whose state is defined as:

$s = (b_1, b_2, p_1, p_2, ac | 0 \leq b_1 \leq B_1; 0 \leq b_2 \leq B_2; 0 \leq p_1 \leq P_1; 0 \leq p_2 \leq P_2; ac = 0 \ \ or \ \ 1)$

Where b_1 and p_1 are the number of processes in the high priority queues; and b_2 and p_2 are the number of processes in the low priority queues; and B_i is the buffer size of the queue i. At time, there is only one high priority queue in the active array and only one low priority queue in the active array, and the remainders are on the expired array. In order to indicate which queues are in these arrays it is used the variable ac, in such a way that if $ac = 0$, then the queues b_1 and b_2 will be in the active array and p_1 and p_2 in the expired array, and when $ac = 1$, vice-versa.

Using standard techniques for the solution of Markov chains, the steady-state probabilities of the CTMC are computed. Again because of the symmetry of the system only the performance measurements associated with the condition $ac = 0$ will be described, i.e., when b_1 and b_2 are in the active array, and p_1 and p_2 are in the expired array. Thus, let $p(b_1, b_2, p_1, p_2, ac)$ be the steady state probability of that Markov model, then the job blocking probability (Pb_i) of a job in the queue i, it is given by the probability of its priority queue is full. Eq. (1) shows, for instance, that probability for the high priority queue in the active array. The

job blocking probability for other arrays may be computed at the same way.

$$Pb_1 = \sum_{b_2=0}^{B_2} \sum_{p_1=0}^{P_1} \sum_{p_2=0}^{P_2} \pi(B_1, b_2, p_1, p_2, 0) \tag{1}$$

The mean delay of the high priority queue and the low priority queue in the active array may be computed as

$$Wb_1 = \frac{\sum_{b1=1}^{B_1} \sum_{b2=0}^{B_2} \sum_{p1=0}^{P_1} \sum_{p2=0}^{P_2} b_1 \pi(b_1, b_2, p_1, p_2, 0)}{\lambda_1(1 - Pb_1)} \tag{2}$$

$$Wb_2 = \frac{\sum_{b1=0}^{B_1} \sum_{b2=1}^{B_2} \sum_{p1=0}^{P_1} \sum_{p2=0}^{P_2} b_2 \pi(b_1, b_2, p_1, p_2, 0)}{\lambda_2(1 - Pb_2)} \tag{3}$$

Where, Pb_2 is the job blocking probability on the low priority queue. Likewise, since, at time, only p_1 and p_2 are in the expired array, the mean delay of the high priority queue and the low priority queue may be, respectively, computed as

$$Wp_1 = \frac{\sum_{b1=0}^{B_1} \sum_{b2=0}^{B_2} \sum_{p1=1}^{P_1} \sum_{p2=0}^{P_2} p_1 \pi(b_1, b_2, p_1, p_2, 0)}{\mu(pr_1 + pr_2)(1 - Pp_1)} \tag{4}$$

$$Wp_2 = \frac{\sum_{b1=0}^{B_1} \sum_{b2=0}^{B_2} \sum_{p1=0}^{P_1} \sum_{p2=1}^{P_2} p_2 \pi(b_1, b_2, p_1, p_2, 0)}{\mu(pr_3 + pr_4)(1 - Pp_2)} \tag{5}$$

Where, Pp_1 and Pp_2 are the job blocking probability on the high and the low priority queue in the expired array. The throughput of the jobs of the high priority queue and the low priority queue in the active array are, respectively, given by:

$$X_1 = qs_1\mu \sum_{b_1>0}^{B_1} \sum_{b_2=0}^{B_2} \sum_{p_1=0}^{P_1} \sum_{p_2=0}^{P_2} \pi(b_1, b_2, p_1, p_2, 0) \tag{6}$$

$$X_2 = qs_2\mu \sum_{b_2>0}^{B_2} \sum_{p_1=0}^{P_1} \sum_{p_2=0}^{P_2} \pi(0, b_2, p_1, p_2, 0) \tag{7}$$

2.1 Reservation model

In this section, an extended model is proposed in order to describe a static reservation allocation policy, where a percentage of the processor capacity (R) is allocated to process one class of applications (Fig. 2). Using this policy, we can divide the capacity of the processor in two variables, R (percentage reserved) used for applications with high priority and $(1 - R)$ for the other applications in the system.

The state of the CTMC of that system is defined as: $s = (b_1, b_2, p_1, p_2, bp, ac | 0 \leq b_1 \leq B_1; 0 \leq b_2 \leq B_2; 0 \leq p_1 \leq P_1; 0 \leq p_2 \leq P_2; 0 \leq b_p \leq B_p; ac = 0 or 1)$.

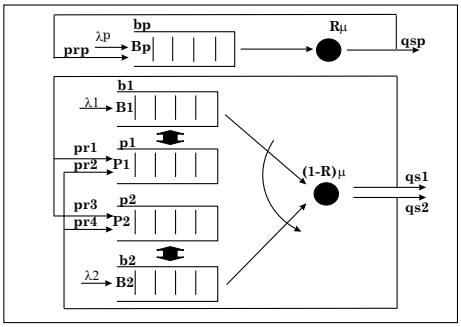


Fig. 2. Resource allocation.

Table 1. Transitions from state $s = (b_1, b_2, p_1, p_2, bp, ac)$ to successor state t for jobs in priority policy.

Successor State	Condition	Rate	Event
$(b_1 + 1, b_2, p_1, p_2, bp, ac)$	$(b_1 < B_1) \wedge (ac = 0)$	λ_1	A job arrives in high priority class
$(b_1, b_2 + 1, p_1, p_2, bp, ac)$	$(b_2 < B_2) \wedge (ac = 0)$	λ_2	A job arrives in low priority class
$(b_1 - 1, b_2, p_1, p_2, bp, ac)$	$(b_1 > 0) \wedge (ac = 0)$	$qs_1(1 - R)\mu$	A job from high class terminates
$(b_1 - 1, b_2, \theta, p_2, bp, ac)$	$(b_1 > 0) \wedge (ac = 0)$ $\begin{cases} \theta = p_1 + 1, \text{ if } p_1 < P_1 \\ \theta = P_1, \quad \text{if } p_1 = P_1 \end{cases}$	$pr_1(1 - R)\mu$	A job is rescheduled to high priority class
$(b_1 - 1, b_2, p_1, \theta, bp, ac)$	$(b_1 > 0) \wedge (ac = 0)$ $\begin{cases} \theta = p_2 + 1, \text{ if } p_2 < P_2 \\ \theta = P_2, \quad \text{if } p_2 = P_2 \end{cases}$	$pr_3(1 - R)\mu$	A job is rescheduled to low priority class
$(b_1, b_2 - 1, p_1, p_2, bp, ac)$	$(b_1 = 0) \wedge (b_2 > 0) \wedge (ac = 0)$	$qs_2(1 - R)\mu$	A job from low class terminates

$(b_1, b_2 - 1, \theta, p_2, bp, ac)$	$(b_1 = 0) \wedge (b_2 > 0) \wedge (ac = 0)$ $\begin{cases} \theta = p_1 + 1, \text{ if } p_1 < P_1 \\ \theta = P_1, \quad \text{if } p_1 = P_1 \end{cases}$	$pr_2(1 - R)\mu$	A job is rescheduled to high priority class
$(b_1, b_2 - 1, p_1, \theta, bp, ac)$	$(b_1 = 0) \wedge (b_2 > 0) \wedge (ac = 0)$ $\begin{cases} \theta = p_2 + 1, \text{ if } p_2 < P_2 \\ \theta = P_2, \quad \text{if } p_2 = P_2 \end{cases}$	$pr_4(1 - R)\mu$	A job is rescheduled to low priority class
$(b_1, b_2, p_1, p_2, bp+1, ac)$	$b_p < B_p$	λ_p	A job arrives in QoS priority class
$(b_1, b_2, p_1, p_2, bp-1, ac)$	$b_p > 0$	$qspR\mu$	A job from QoS class terminates
$(b_1, b_2, p_1, p_2, bp-1, ac)$	$b_p > 0$	$prpR\mu$	A job is rescheduled, but before it is decremented
$(b_1, b_2, p_1, p_2, bp+1, ac)$	$b_p < B_p$	$prpR\mu$	A job is rescheduled, but after it is incremented
$(b_1, b_2, p_1, p_2, bp, ac+1)$	$(ac = 0) \wedge ((b_1 = 0) \wedge (b_2 = 0)) \wedge ((p_1 > 0) \vee (p_2 > 0))$	mtv	Change of arrays, b_1 and b_2 become expired
$(b_1, b_2, p_1, p_2, bp, ac-1)$	$(ac = 1) \wedge ((p_1 = 0) \wedge (p_2 = 0)) \wedge ((b_1 > 0) \vee (b_2 > 0))$	mtv	Change of arrays, b_1 and b_2 become active

Transitions from state s to all possible successor states are reported in Table 1 along with their rates and conditions under which the transitions exist; the last column indicates the type of event to which a transition refers. When $ac = 0$, if a job is generated in the high priority queue in the active array, the occupancy of that queue, b_1, will increase by one unit. A rescheduled job from that queue will go to the high priority queue in the expired array with rate $pr_1(1 - R)$ or to the low priority queue in the expired array with rate $pr_3(1 - R)$. In the first case the job keeps the same priority and, in the latter, the priority is decreased. A job can leave the high priority queue in the active array, after finishing its processing with rate $qs_1(1 - R)$. An arrival in the low priority queue in the active array takes place with rate and increases b_2 by one unit.

Since the system under analysis is finite, when a buffer (active or expired arrays) is full an incoming or rescheduled job is blocked. After switching, the queues that were in the expired array (p_1 and p_2) become active and vice-versa. The variable bp represents parallel jobs. We assumed that $mtv = 10^{-6}$. The sys-

tem is symmetric, which makes quite natural the match of the other transitions of the model.

The variable bp represents parallel jobs. We assume that $mtv = 10^{-6}$. The system is symmetric, which makes quite natural the understanding of the Table 1.

Due to the lack of space and for simplicity only the performance measurements of the high priority jobs (parallel) that demand QoS guarantees are presented. Assuming that $ac = 0$, the mean delay perceived by that processes are computed as

$$W_{pb} = \frac{\sum_{b_1=0}^{B_1}\sum_{b_2=0}^{B_2}\sum_{p_1=0}^{P_1}\sum_{p_2=0}^{P_2}\sum_{b_p=1}^{B_P} b_p\pi(b_1,b_2,p_1,p_2,b_p,0)}{(\lambda_p + prpR\mu)(1-Pb_p)} \tag{8}$$

Where Pb_p is blocking probability of high priority jobs that demand QoS guarantees derived as Eq.(1). The throughput is given by:

$$X_{pb} = qspR\mu \sum_{b_1=0}^{B_1}\sum_{b_2=0}^{B_2}\sum_{p_1=0}^{P_1}\sum_{p_2=0}^{P_2}\sum_{b_p>0}^{B_P} \pi(b_1,b_2,p_1,p_2,b_p,0) \tag{9}$$

3 Performance study

In this section some numerical results are presented to evaluate how adequate is the Markov model to scheduling GNU/Linux with and without resource allocation policy. First, we present the performance of the Linux Markovian model. For validation purpose, Linux scheduler was simulated by using an academic version of a powerful tool named ARENA©[9]. Some measures were obtained through system calls which collect data for later analysis, minimizing the overhead in kernel (this can be obtained in www.lprad.ufpa.br/parqos). Table 2 summarizes the parameters used.

Table 2. Input data.

High Priority	Measures	Low Priority:	Measures
λ_1	7	λ_2	7,3
pr_1	0,1	pr_2	0
pr_3	0,09	pr_4	0,67
Avarage Buffer	5	Avarage Buffer	5

To validate the probability distributions adopted, models use input data obtained from the real system. In these data, Kolmogorov-Smirinov (K-S) goodness of fit tests were applied, using the trial version of BestFit©tool [10].

These data were used as parameters of probability distributions in question (Poisson for inter-arrivals times). The simulation results were collated with the performance measures obtained from the real system. As the numerical results of that comparison match (very similar), the values may be considered validated for the analytical model.

To implement CPU allocation policy it is important to study the CPU behavior. Assuming the table above, it represents a situation where scheduler is very busy and the inputs are Poisson traffic. A new application is added in λ_1, simulating a situation of great workload. Table 3 ilustrates the Markovian model output. As expected, higher the traffic load, bigger the throughput and, for that reason, longer the mean waiting time, longer is the blocking probability. In the table, 0% represents the system behavior performance with just λ_1 and λ_2. λ_p is derived from λ_1(5%, 10%, 20%, 30%, 40%) and represents the impact of adding an application to the system.

Table 3. Performance measurements.

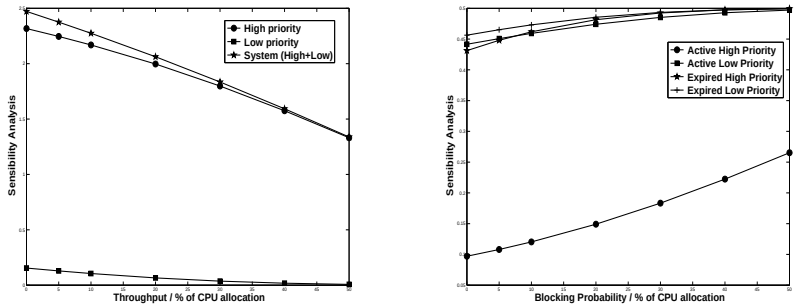
Queue Waiting Time						
	0%	5%	10%	20%	30%	40%
Active High Priority	0,21246	0,21491	0,21687	0,21943	0,22037	0,21995
Active Low Priority	0,59056	0,60058	0,60970	0,62533	0,63785	0,64774
Expired High Priority	6,05108	6,28370	6,48294	6,79442	7,01313	7,16437
Expired Low Priority	0,85739	0,87383	0,88871	0,91386	0,93334	0,94802
Blocking Probability						
	0%	5%	10%	20%	30%	40%
Active High Priority	0,09701	0,10763	0,11827	0,13925	0,15940	0,17838
Active Low Priority	0,44159	0,44832	0,45431	0,46434	0,47214	0,47817
Expired High Priority	0,43127	0,44343	0,45346	0,46847	0,47854	0,48530
Expired Low Priority	0,45635	0,46216	0,46734	0,47591	0,48241	0,48723

The investigation of the impact on increasing CPU allocation (sensibility analysis) is interesting because it shows the system behavior that determines at which extent the CPU is efficiently and fairly used by all processes. The data in which this analysis was conducted is described in Table 2, adding a load of 50% to λ_1 (originated by the parallel application).

The Fig. 3.a shows the processes active high priority (parallel processes), active low priority and expired low priority remain with values Queuing Waiting Time almost constant with respect to several CPU allocation (from 0 to 50%). This implies that the impact on the waiting time is low, for the processes high priority, active low priority and expired low priority, with respect to the increase in the CPU allocation. On the other hand, the expired high priority processes have their time substantially increased when the CPU allocation gets larger. (for instance, 6 seconds of waiting to 0% of CPU allocation and 15 seconds to 50% of CPU allocation)

The blocking probability presents their worst values for active low priority, expired high priority and expired low priority (around 0.5 for 50% of CPU allocation) (Fig. 3.b). The smallest blocking probability is related to the active high priority processes.

Throughput is reduced approximately by 50% for both high priority and low priority processes (considering 0 to 50% of CPU allocation) (Fig. 4).



(a) (b)

Fig. 3. (a) Queuing Waiting Time / % of CPU Allocation (b)Blocking Probability / % of CPU Allocation.

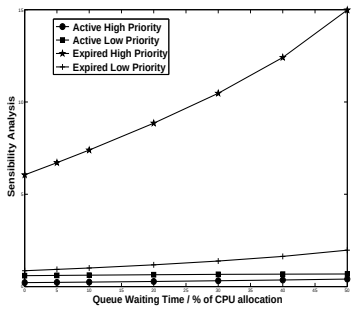


Fig. 4. Throughput / % of CPU Allocation.

4 Final remarks

In this paper, a Markovian Linux scheduler model has been presented and proposed for performance study and, in addition, an extended model, which uses

static resource allocation policy. Through the sensibility analysis, it has been concluded that the performance of the parallel applications with QoS are greatly improved in its throughput. However, others applications have suffered some limitations. The contributions of this paper are: (1) Proposal of performance models for GPOS scheduler; (2) Proposal of a resource (CPU) allocation scheme in a parallel computing environment as well as showing through numerical results, obtained from its Markovian model, improvement of performance of parallel applications when compared to other applications.

Currently, we are implementing another extended model which uses dynamic CPU allocation policy. As future work, we are performing experiments with Markov decision process to find optimal admission control and scheduling strategies aiming at improving the resource (CPU and memory) allocation for parallel applications.

This work is supported by CNPq and CAPES.

References

1. Zhang, Y., Sivasubramaniam, A., Moreira, J., Franke, H.: Impact of Workload and System Parameters on Next Generation Cluster Scheduling Mechanisms. IEEE Transactions on Parallel and Distributed Systems, Vol. **12** (2001) 967-985.
2. Hwang, K., Xu, Z.: Scalable Parallel Computing - Technology, Architecture and Programming, WCB/ McGraw-Hill, (1998).
3. Niyato, D., Hossain, E.: Analysis of Fair Scheduler and Connection Admission Control in Differentiated Services Wireless Networks. IEEE International Conference on Communications, Vol. **5** (2005) 3137 - 3141.
4. Carvalho, G., Rodrigues, R., Francs, C., Costa, J., Carvalho, S.: Modelling and Performance Evaluation of Wireless Networks. Lecture Notes in Computer Science, Vol. **3124**. Heidelberg Germany (2004) 595-600.
5. Manolache, S., Eles, P., Peng, Z.: Schedulability Analysis of applications with Stochastic Task Execution Times. ACM Transactions on Embedded Computing Systems, Vol. **3**. November (2004) 706-735.
6. Chanin, R., Corrêa, M., Fernandes, P., Sales, A., Scheer, R., Zorzo, A.F.: Analytical Modeling for Operating System Schedulers on NUMA Systems, in Proc. of the 2nd International Workshop on Practical Applications of Stochastic Modelling, PASM05, University of Newcastle upon Tyne, UK, July (2005).
7. Love, R.: Linux Kernel Development, SAMS, 1st edn., (2003).
8. Wei, W., Wang, B., Towsley, D.: Continuous-Time Hidden Markov Models for Network Performance Evaluation, Performance Evaluation, Vol.**49**, (2002), pp. 129-146.
9. Rockwell Automatation - www.arenasimulation.com, accessed in 02/15/2006.
10. Palisade - www.palisade.com/bestfit, accessed in 02/18/2006.