Um Modelo de Desempenho Markoviano para Escalonamento de Processos Paralelos no GNU/Linux

Regiane Y. Kawasaki[1], Luiz A. Guedes[2], Carlos R. L. Frances[1], João C. W. A. Costa[1], Glaucio H. S. Carvalho[1], Diego L. Cardoso[1], Marcelino S. Silva[1], Luiz D. C. Augusto[1]

[1]Laboratório de Planejamento de Redes de Alto Desempenho – Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) – Universidade Federal do Pará (UFPA)
CP 476 – 66.075-110 – Belém – PA – Brasil

[2]Departamento de Automação e Computação (DCA) – Universidade Federal do Rio Grande do Norte (UFRN)
59072-970 – Natal – RN – Brasil

{kawasaki,diego,rfrances,weyl,ghsc,marcelino,ldaugusto}@ufpa.br; affonso@dca.ufrn.br

Abstract. *Parallel Computing become a powerful tool to overcome certain types of computational problems in many areas such as engineer, especially due to the increasing diversity of platforms for the execution of this type of application. The use of parallel computing over LANs (Local Area Networks) is an alternative in the universe of parallel machines, but needs to imply QoS (Quality of Service) parameters so 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 GPOS (General Purpose Operating Systems).*

Resumo. *A computação Paralela é uma poderosa ferramenta na solução de problemas computacionais em várias áreas tais como engenharia, especialmente pelo aumento da diversidade de plataformas para a execução deste tipo de aplicações. O uso da computação paralela em LANs (Redes Locais) é uma alternativa no universo de máquinas paralelas, entretanto necessita impor parâmetros de QoS (Qualidade de Serviço) para que sejam executadas de forma eficiente. Neste cenário, o desenvolvimento de esquemas de alocação de recursos possuem um papel muito importante para satisfazer os requisitos de QoS das aplicações paralelas. Neste artigo é proposto e apresentado modelos markovianos para esquemas de alocação de recursos (alocação de CPU) em GPOS (Sistemas Operacionais de Propósito Geral).*

1. Introdução

Aplicações *soft real time*, que utilizam CPU periodicamente, vêm se tornando uma grande fatia dos programas que são executados em máquinas com sistemas operacionais de propósito geral (*General Purpose Operating Systems* - GPOS), tais como Gnu/Linux e Windows 2003 Server. Principalmente em servidores que prestam serviços a instituições e empresas. Aplicações típicas dessa classe incluem simulações,

streaming de áudio e vídeo, serviços tais como servidores web (IIS ou Apache), servidores de nomes, entre outros. Não obstante, essas aplicações normalmente não podem ser classificadas como aplicações *hard real time*, entretanto precisam ser executadas em um intervalo de tempo pertinente e utilizando uma grande quantidade dos recursos da máquina.

A implementação de técnicas de QoS (*Quality of Service* - Qualidade de Serviço) em GPOS tem sido uma prática bastante difundida procurando satisfazer as necessidades existentes nessas aplicações. Entre as técnicas desenvolvidas, a implementação de algoritmos de escalonamento que possibilitam um escalonamento diferenciado tem se destacado nas pesquisas atuais.

Para implementar alocação de recursos, pode-se, por exemplo, implementar um controle de admissão (CAC) conjuntamente com o processo de escalonamento, de forma que seja possível detectar-se as necessidades das aplicações e ajustar o sistema de maneira a prover o serviço requisitado, quando possível. Este tipo de alocação de recursos tem sido estudado na literatura, entretanto não voltado para GPOS e aplicações sobre o mesmo. Por exemplo, em [Niyato e Hossain 2005] os autores propuseram um escalonador justo (*fair scheduler*) e um controle de admissão para serviços diferenciados em redes móveis sem fio. Esta proposta utiliza duas filas, sendo uma para aplicações de QoS e outra para aplicações de "melhor esforço". Utilizando um modelo markoviano os autores geraram algumas medidas e analisaram algumas interdependências. Assim como em [Meo e Marsan 2004], onde é apresentado um estudo analítico baseado na implementação de reserva de recursos em redes celulares GSM/GPRS.

Via de regra, quando se pretende otimizar ou propor modificações em sistemas complexos, propõem-se modelos que representem sistemicamente o fenômeno estudado e, a partir desses modelos, pode-se fazer uma série de inferências que norteiem decisões de projeto e proponham alternativas a possível deficiências do sistema em estudo. Este artigo propõe e apresenta um modelo markoviano para o escalonamento do GNU/Linux tradicional e a partir desse escalonamento sugere um esquema alternativo de alocação de recursos para tratamento diferenciado de processos paralelos em sistemas GNU/Linux.

Este artigo está organizado como segue. A seção 2 descreve a funcionalidade do escalonador atualmente utilizado em um GPOS. A Seção 3 apresenta o modelo analítico do escalonador utilizado atualmente em um GPOS e após é proposto um novo escalonador que através de algumas alterações no código do GPOS, possibilitando a implementação da política de reserva de recursos. Na seção 4 são apresentados alguns resultados numéricos obtidos do modelo matemático com e sem reserva de recursos. Finalmente na seção 5 são realizados alguns comentários finais deste trabalho.

2. Escalonador

O escalonador do GNU/Linux é multitarefa preemptivo (*preemptive multitasking*). Para definir quem deverá ser processado, o escalonador trabalha com uma política (*policy*), através da qual poderá otimizar a utilização do processador. A política adotada pelo *kernel* 2.6 é de prioridades, ou seja, aqueles com maiores prioridades serão executados primeiro. Esta prioridade é dinâmica, podendo aumentar ao diminuir durante a execução do processo.

A forma na qual o escalonamento dos processos é realizada é totalmente diferenciada de como era realizado nas versões do *kernel* anteriores, pode-se exemplificar de forma sucinta [Damásio, 2004]:

Tabela 1. *Kernel* 2.4 Vs. *Kernel* 2.6

Escalonador do kernel 2.4:	***Escalonador do kernel 2.6:***
Para (cada processo no sistema) *{* *Ache o valor de "merecimento" (prioridade dinâmica) do processo;* *Se (esse é o processo com maior "merecimento" (ou seja, maior prioridade))* *{* *Lembre-se desse processo* *}* *}* *Rode o processo com maior "merecimento";*	*Vá para a fila com maior prioridade;* *Pegue o primeiro processo nessa fila;* *Rode esse processo;*

A estrutura básica do escalonador é a fila de execução (*struct runqueue*). Esta fila está definida dentro do arquivo "sched.c", sendo responsável por conter todos os processos executáveis para um determinado processador. Qualquer processo executável tem de estar em uma fila de execução e cada processador possui a sua *struct runqueue*. Para se obterem filas de execução específicas e trabalhar com elas, são utilizados macros que apontam para essas filas. Por exemplo, a macro *cpu_rq* aponta para a fila de execução do processador; já a macro *this-rq* aponta para a fila de execução atual [Love, 2004].

Cada fila de execução possui dois vetores (*arrays*) de prioridade: um vetor ativo e outro expirado, definidos também no "sched.c". Cada vetor de prioridade contém uma lista de todos os processos executáveis, onde cada fila possui uma prioridade estática e é ordenada de acordo com essa prioridade. Através de um "mapa de bits" é listada a existência ou não de processo executáveis em uma determinada fila e assim os processos com maior prioridade são encontrados com maior eficiência. A procura pelo processo de maior prioridade se restringe a esse "mapa de bits", que utiliza um algoritmo de procura chamado *sched_find_first_bit()*, o qual procura o primeiro elemento um ("1"), e informa a existência de um ou mais processos para execução com aquela determinada prioridade. Após terminar sua execução, ou por finalizar a tarefa ou acabar seu tempo de execução, o tempo e a prioridade são recalculados. Recolocando o processo em sua determinada fila (baseada na prioridade) em outro vetor, chamado vetor expirado. Após a execução de todos os processos do vetor ativo, os vetores são trocados, onde o que era ativo passa a ser expirado e vice-versa.

Resumidamente, o vetor ativo referencia aqueles processos executáveis que ainda possuem *timeslice* e o vetor expirado possui todas as tarefas que já utilizaram todo seu *timeslice*. Quando uma tarefa utiliza toda sua fatia de tempo, esta tarefa tem seu *quatum* recalculado e então é movida para a o vetor expirado. Quando o vetor ativo não possui mais tarefas com fatias de tempo restante, os vetores são trocados.

Objetivando ter-se um entendimento sistêmico do escalonamento de processos do GNU/Linux, é possível modelar esse escalonamento utilizando-se uma técnica apropriada de modelagem de sistemas complexos. Algumas técnicas relatadas na literatura especializada são: redes de fila, redes de Petri e cadeias de Markov. A partir do modelo gerado e das medidas obtidas podem ser tomadas decisões de projeto, visando à otimização do escalonamento

3. Modelo Analítico

Como contribuição ao estudo e entendimento do escalonamento de processos do GNU/Linux, nesta seção, o *kernel* 2.6 foi modelado com dois vetores: um vetor ativo e outro expirado, sendo que cada vetor possui duas filas que adotam a política de escalonamento por prioridade para as filas e, em cada fila, utiliza-se a política FIFO. Obviamente o modelo apresentado pode facilmente ser estendido para "n" filas. Finalmente, são apresentadas algumas modificação realizadas no modelo a fim de inserir o conceito de alocação de recursos com QoS.

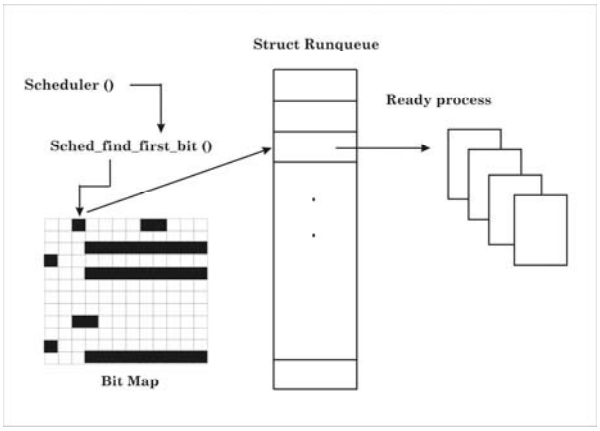

Figura 1. Escalonador Linux

O escalonador do GNU/Linux pode ser modelado através de duas filas comuns ao servidor (processador). Duas classes de clientes entram no sistema. Clientes da primeira classe representam processos de alta prioridade, tais como processos do *kernel* (atualizações, geração de processos, etc). Estes clientes entram no sistema de acordo com uma distribuição de Poisson com parâmetro λ_1 e solicitam o processador com um tempo de serviço representado por uma exponencial negativa, com taxa μ. A segunda classe de clientes representa os processos de baixa prioridade, como compiladores, *browsers*, aplicações paralelas e todas as demais aplicações. Como anteriormente dito, os clientes são gerados d acordo com uma distribuição poissoniana com parâmetro λ_2 e solicitam o processador com um tempo de serviço representado por uma exponencial negativa, com taxa μ. Para validar as distribuições de probabilidade adotadas, foram simulados os modelos com os dados de entrada obtidos a partir do sistema real, os quais foram utilizados como parâmetros das distribuições em questão (Poisson para chegada e exponencial negativa para atendimento de processos). O resultado da simulação foi confrontado com as medidas de desempenho obtidas também no sistema real. Pela aproximação obtida nessa comparação, os valores foram considerados validados para o modelo analítico.

Todos os *jobs* possuem as mesmas características e podem ser servidos e armazenados no vetor ativo que possui capacidade igual a B (B_1 para a fila de alta prioridade e B_2 para a de baixa prioridade).

Um *job* é processado caso o processador esteja disponível. O *job* é removido do vetor ativo em duas condições: (a) quando o *job* termina seu processamento, com probabilidade qs_1 (para processos de alta prioridade) e qs_2 para os outros processos; ou (b) quando precisa ser re-escalonado (os processos vão para o vetor expirado e seu *time-slice* e prioridade são recalculados). Quando o processo é oriundo de uma fila de prioridade alta, pode ser escalonado com probabilidade pr_1 (para a mesma classe de processos) ou pr_3 para a classe de baixa prioridade. Quando um processo se origina de uma fila de menor prioridade, pode ser escalonado com probabilidade pr_2 para uma classe de maior prioridade ou com probabilidade pr_4 para a mesma classe de prioridade. Um *job* de menor prioridade só é processado quando todos os processos de maior prioridade forem executados. Quando as filas estão vazias no vetor ativo, os vetores são trocados, ou seja, o vetor ativo passa a ser expirado e vice-versa (conforme dinâmica apresentada na Figura 2). Esta troca possui um tempo associado, entretanto é muito baixo e pode ser desprezado no estudo.

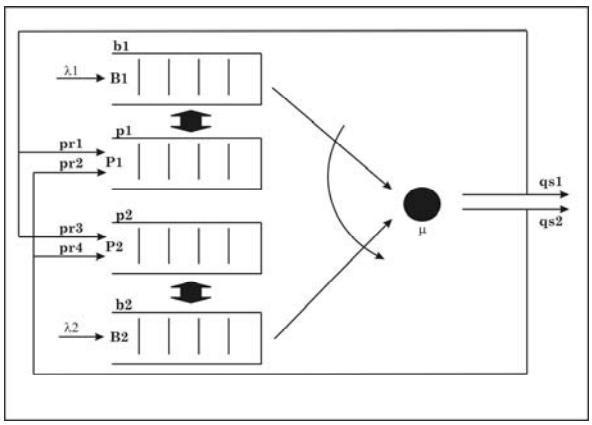

Figura 2. Modelo do Sistema.

Assumindo o que foi dito anteriormente, foi desenvolvido um modelo de uma cadeia de Markov contínua (CTMC) do sistema, sendo os estados definidos como $s = (b_1, b_2, p_1, p_2 / 0 \le b_1 \le B_1; 0 \le b_2 \le B_2;\ 0 \le p_1 \le P_1; 0 \le p_2 \le P_2)$, onde: b_1 representa o número de processos na fila de alta prioridade no vetor ativo; b_2 representa o número de processos na fila de menor prioridade no vetor ativo; p_1 representa o número de processos na fila de alta prioridade no vetor expirado; p_2 representa o número de processos na fila de baixa prioridade no vetor expirado. E quando os vetores são trocados: p_1 representa o número de processos na fila de alta prioridade no vetor expirado; p_2 representa o número de processos na fila de baixa prioridade no vetor expirado; b_1 representa o número de processos na fila de alta prioridade no vetor ativo; b_2 representa o número de processos na fila de baixa prioridade no vetor ativo;

Sendo S o espaço de estados da cadeia de Markov modelada. O número de estados em S é igual a $(B_1 + 1)(B_2 + 1)(P_1 + 1)(P_2 + 1)$. Mudanças das variáveis de estado b_1 e b_2 são: acréscimo quando um *job* é criado, decremento quando um *job* termina ou é

re-escalonado no sistema. Quando um *job* é gerado, a ocupação dos *buffers* B_1 ou B_2 são incrementadas de 1. Quando não existem posições livres no buffer, os *jobs* são bloqueados e perdidos. Quando os *jobs* são re-escalonados, p_1 e p_2 são incrementados baseando-se na decisão do escalonador, se o *job* vier de uma fila de alta prioridade, ele pode manter sua prioridade e incrementar p_1 ou incrementar p_2 e diminuir de prioridade. Quando um *job* vem de uma fila de baixa prioridade, ele pode incrementar p_1 se esse *job* mudar de classe e aumentar a prioridade ou incrementar p_2 se o *job* mantém sua prioridade. Finalmente, um *job* é processado com sucesso proveniente de uma fila de alta prioridade com taxa $qs_1\mu$, e $qs_2\mu$ quando vier de uma fila de menor prioridade. Este processo de escalonamento funciona da mesma forma quando p_1 e p_2 forem os vetores ativos e b_1 e b_2 forem os expirados.

Utilizando técnicas tradicionais de solução de cadeias de Markov, podem-se computar as probabilidades-limite da CTMC. Admitindo-se que π(s) é vetor do estado estacionário (será usada a notação $\pi\ (b_1, b_2, p_1, p_2)$ para π(s)). A partir de π(s), foram obtidas e computadas algumas medidas de desempenho interessantes. A probabilidade de bloqueio das filas 1 e 2 podem ser calculadas da seguinte forma:

Quando b_1 e b_2 forem vetores ativos:

$$Pb_1 = \sum_{b_1=B_1,\, b_2} \pi_i \ \forall \mathrm{i} \in \mathrm{S} \qquad (1) \qquad Pb_2 = \sum_{b_1,\, b_2=B_2} \pi_i \ \forall \mathrm{i} \in \mathrm{S} \qquad (2)$$

Então, a probabilidade de bloqueio média pode ser obtida por:

$$Pb_i = \sum_{b_i=B_i,\, \forall\, k \neq i \in S} \pi_j \ \forall \mathrm{i} \in \mathrm{S} \qquad (3)$$

O atraso médio para cada buffer:

$$W_i = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2} b_1\pi(b_1, b_2, p_1, p_2)}{(1-Pb_1)(\lambda_1 + (pr_1 + pr_2)\mu)} \qquad (4) \qquad W_2 = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2} b_2\pi(b_1, b_2, p_1, p_2)}{(1-Pb_2)(\lambda_2 + (pr_3 + pr_4)\mu)} \qquad (5)$$

E finalmente, a utilização de CPU por:

$$U = 1 - \pi_0 \qquad (6)$$

Onde π_0 é a probabilidade de estar no estado de ociosidade.

Quando b_1 e b_2 forem vetores expirados:

$$Pb_1 = \sum_{p_1=P_1,\, p_2} \pi_i \ \forall \mathrm{i} \in \mathrm{S} \qquad (7) \qquad Pb_2 = \sum_{p_1,\, p_2=P_2} \pi_i \ \forall \mathrm{i} \in \mathrm{S} \qquad (8)$$

A fórmula geral para a probabilidade de bloqueio media é obtida por:

$$Pb_i = \sum_{p_i=P_i,\, \forall\, k \neq i \in S} \pi_j \ \forall \mathrm{i} \in \mathrm{S} \qquad (9)$$

O atraso médio para cada buffer:

$$W_i = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2} p_1\pi(b_1, b_2, p_1, p_2)}{(1-Pb_1)(\lambda_1 + (pr_1 + pr_2)\mu)} \qquad (10) \qquad W_2 = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2} p_2\pi(b_1, b_2, p_1, p_2)}{(1-Pb_2)(\lambda_2 + (pr_3 + pr_4)\mu)} \qquad (11)$$

3.1. Modelo de Alocação de Recursos

O atual escalonador é uma solução efetiva para sistemas operacionais de propósito geral (GPOS), mas para situações em que é necessário que uma aplicação

específica tenha um tratamento diferenciado em relação às demais, baseado em alocação de recursos (CPU, memória, disco, etc), o escalonador do GNU/Linux não está preparado. Objetivando atender a esta situação, algumas modificações no atual escalonador ou um novo escalonador podem ser possíveis soluções.

Nesta seção é descrito um modelo estendido de forma a prover a política de alocação estática de recursos, onde um percentual da capacidade do processador (R) é alocado para uma determinada classe de aplicações. Desta forma, os processos dessa classe irão utilizar este percentual reservado de CPU. Assim, com esta política, pode-se dividir a capacidade do processador em duas variáveis, *R* (percentual reservado) utilizado para aplicações com QoS, normalmente as de maior prioridade e *1-R* para todas as outras aplicações do sistema. A CTMC que descreve este sistema possui um estado a mais que o modelo anterior e, com isso, novas transições de estados são adicionadas. Além disso, as transições modificam com a adição da variável (*R*). A cardinalidade do espaço de estados obtém um acréscimo de um estado a mais $(B_1+1)(B_2+1)(P_1+1)(P_2+1)(BP+1)$ e conseqüentemente algumas novas condições são utilizadas (Tabela 2) juntamente com estado sucessor, taxas e o evento que a transferência se refere.

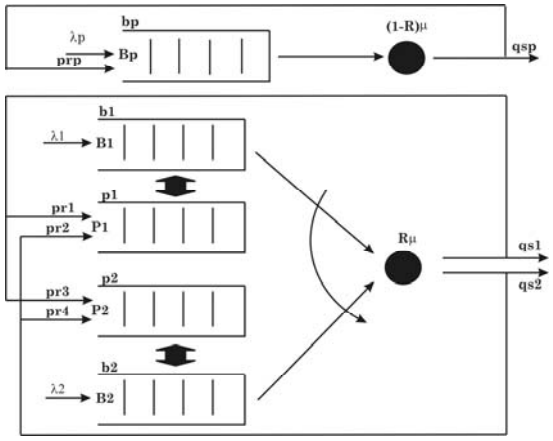

Figura 3. Alocação de recursos

Tomando-se como base estas novas probabilidades de transição, podem-se computar as probabilidades do estado estacionário da CTMC, e para este novo π(s) podem-se obter medidas de desempenho como no modelo anterior, exceto pelo atraso médio, que neste modelo é obtido da seguinte forma:

Quando b_1 e b_2 são vetores ativos:

$$W_1 = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2}\sum_{bp} b_1\pi(b_1,b_2,p_1,p_2bp)}{(1-Pb_1)(\lambda_1+(pr_1+pr_2)\mu+(pr_1+pr_2)(1-R)\mu)} \quad (12)$$

$$W_2 = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2}\sum_{bp} b_2\pi(b_1,b_2,p_1,p_2bp)}{(1-Pb_2)[\lambda_2+(pr_3+pr_4)\mu+(pr_3+pr_4)(1-R)\mu)} \quad (13)$$

$$W_p = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2}\sum_{bp} bp\pi(b_1,b_2,p_1,p_2bp)}{(1-Pb_2)(\lambda_2+(prp)R\mu)} \quad (14)$$

Quando b_1 e b_2 são vetores expirados:

$$W_1 = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2}\sum_{bp} p_1\pi(b_1,b_2,p_1,p_2bp)}{(1-Pb_1)(\lambda_1+(pr_1+pr_2)\mu+(pr_1+pr_2)(1-R)\mu)} \quad (15)$$

$$W_2 = \frac{\sum_{b1}\sum_{b2}\sum_{p1}\sum_{p2}\sum_{bp} p_2\pi(b_1,b_2,p_1,p_2bp)}{(1-Pb_2)[\lambda_2+(pr_3+pr_4)\mu+(pr_3+pr_4)(1-R)\mu)} \quad (16)$$

Tabela 2. Transições da configuração $s = (b_1, b_2, p_1, p_2, bp)$ para a sucessora para *jobs* com prioridade.

Successor State	Condition	Rate	Event
$(b_1+1, b_2, p_1, p_2, bp)$	$b_1 < B_1$	λ_1	Chegada de um *job* de alta prioridade
$(b_1, b_2+1, p_1, p_2, bp)$	$b_2 < B_2$	λ_2	Chegada de um *job* de baixa prioridade
$(b_1-1, b_2, p_1, p_2, bp)$	$b_1 > 0$	$qs_1(1-R)\mu$	Término de um *job* de alta prioridade
$(b_1-1, b_2, \theta, p_2, bp)$	$b_1 > 0$ $\begin{cases}\theta = p_1+1, if\ p_1 < P_1\\ \theta = P_1, if\ p_1 = P_1\end{cases}$	$pr_1(1-R)\mu$	Re-escalonamento de um *job* para alta prioridade
$(b_1-1, b_2, p_1, \theta, bp)$	$b_1 > 0$ $\begin{cases}\theta = p_2+1, if\ p_2 < P_2\\ \theta = P_2, if\ p_2 = P_2\end{cases}$	$pr_3(1-R)\mu$	Re-escalonamento de um *job* para baixa prioridade
$(b_1, b_2-1, p_1, p_2, bp)$	$(b_1 = 0) \wedge (b_2 > 0)$	$qs_2(1-R)\mu$	Término de um *job* de baixa prioridade
$(b_1, b_2-1, \theta, p_2, bp)$	$(b_1 = 0) \wedge (b_2 > 0)$ $\begin{cases}\theta = p_1+1, if\ p_1 < P_1\\ \theta = P_1, if\ p_1 = P_1\end{cases}$	$pr_2(1-R)\mu$	Re-escalonamento de um *job* para alta prioridade
$(b_1, b_2-1, p_1, \theta, bp)$	$(b_1 = 0) \wedge (b_2 > 0)$ $\begin{cases}\theta = p_2+1, if\ p_2 < P_2\\ \theta = P_2, if\ p_2 = P_2\end{cases}$	$pr_4(1-R)\mu$	Re-escalonamento de um *job* para baixa prioridade
$(b_1, b_2, p_1+1, p_2, bp)$	$p_1 < P_1$	λ_1	Chegada de um *job* de alta prioridade
$(b_1, b_2, p_1, p_2+1, bp)$	$p_2 < P_2$	λ_2	Chegada de um *job* de baixa prioridade
$(b_1, b_2, p_1-1, p_2, bp)$	$p_1 > 0$	$qs_1(1-R)\mu$	Término de um *job* de alta prioridade
$(\theta, b_2, p_1-1, p_2, bp)$	$p_1 > 0$ $\begin{cases}\theta = b_1+1, if\ b_1 < B_1\\ \theta = B_1, if\ b_1 = B_1\end{cases}$	$pr_1(1-R)\mu$	Re-escalonamento de um *job* para alta prioridade
$(b_1, \theta, p_1-1, p_2, bp)$	$p_2 > 0$ $\begin{cases}\theta = b_2+1, if\ b_2 < B_2\\ \theta = B_2, if\ b_2 = B_2\end{cases}$	$pr_3(1-R)\mu$	Re-escalonamento de um *job* para baixa prioridade
$(b_1, b_2, p_1, p_2-1, bp)$	$(p_1 = 0) \wedge (p_2 > 0)$	$qs_2(1-R)\mu$	Término de um *job* de baixa prioridade
$(\theta, b_2, p_1, p_2-1, bp)$	$(p_1 = 0) \wedge (p_2 > 0)$	$pr_2(1-R)\mu$	Re-escalonamento de um *job* para alta prioridade

	$\begin{cases}\theta = b_1 + 1, if\ b_1 < B_1 \\ \theta = B_1, if\ b_1 = B_1\end{cases}$		
	$(p_1 = 0) \wedge (p_2 > 0)$		
$(b_1, \theta, p_1, p_2 - 1, bp)$	$\begin{cases}\theta = b_2 + 1, if\ b_2 < B_2 \\ \theta = B_2, if\ b_2 = B_2\end{cases}$	$pr_4(1-R)\mu$	Re-escalonamento de um *job* para baixa prioridade
$(b_1, b_2, p_1, p_2, bp + 1)$	$b_p < B_p$	λ_p	Chegada de um *job* de com reserva de recursos
$(b_1, b_2, p_1, p_2, bp - 1)$	$b_p > 0$	$prpR\mu$	Término de um *job* com reserva de recursos
(b_1, b_2, p_1, p_2, bp)	$b_p \leq B_p$	$qspR\mu$	Re-escalonamento de um *job* com reserva de recursos

4. Avaliação de Desempenho

4.1. Protótipo Par_QoS

O modelo acima foi idealizado para trabalhar como parte da arquitetura Par-QoS [Kawasaki et al 2003]. Esta arquitetura combina os conceitos de processamento paralelo, redes locais (LAN) e de GPOS. Possui algumas motivações e justificativas, que estão diretamente relacionadas à não viabilidade da infra-estrutura de comunicação disponíveis. Nestes cenários, soluções tais como *grids* computacionais [Foster et al. 2001] possuem sua aplicabilidade restrita.

Para garantir as características desejadas, a arquitetura Par_QoS utiliza o conceito de QoS fim-a-fim, basicamente em dois níveis, nas redes locais e nos GPOS. No primeiro, os pacotes provenientes das aplicações paralelas são roteados com uma prioridade superior aos outros pacotes da rede, utilizando o conceito *diffserv* (serviços diferenciados), no qual os pacotes são marcados utilizando o campo ToS (*Type of Service*) no datagrama TCP/IP. Fluxos são divididos em classes e cada classe possui uma prioridade de roteamento em relação aos outras classes.

No nível de GPOS, o kernel foi modificado para prossibilitar a alocação de recursos (neste caso, capacidade de processamento) para aplicações específicas. Entre os existentes, o GNU/Linux é o que se destaca por possuir código aberto, não necessidade de licenças e ampla utilização. Um protótipo já foi desenvolvido e está sendo amplamente utilizado na Universidade Federal do Pará. Baseado neste protótipo, o nível de GPOS foi monitorado para tornar possível validar, baseado em resultados numéricos, o modelo markoviano gerado.

4.2. Resultados Numéricos

Nesta seção serão apresentados alguns resultados numéricos obtidos para avaliar a adequação do modelo de escalonamento markoviano do GNU/Linux e para o modelo que utiliza a política de alocação de recursos. Primeiramente é apresentado o modelo de desempenho do GNU/Linux. Para fins de validação, o GNU/Linux foi modelado utilizando a ferramenta de simulação ARENA®. Alguns dados foram obtidos através de chamadas de sistemas especialmente desenvolvidas para este propósito, que coletaram

dados sem gerar *overhead* no *kernel* (maiores informações podem ser obtidas em www.lprad.ufpa.br/parqos)

Os valores coletados no sistema real foram introduzidos no modelo e representam uma situação na qual a CPU está sendo largamente utilizada. Para obter esses valores, um *benchmark* foi criado utilizando a recompilação do *kernel*, com as mesmas opções selecionadas por 100 vezes, em três computadores diferentes (dois Atholn XP 2.6 com 512Mb de RAM e um Athlon XP 2.8 com 512 Mb de RAM). Dados obtidos pelas chamadas de sistemas foram inseridos em ambos os casos (no modelo markoviano e na simulação utilizando o ARENA®), a partir dos quais as saídas obtidas foram comparadas e mostraram o mesmo comportamento nas variáveis utilização *de CPU* e *tempo de espera em fila*. A Tabela 3 sumariza os parâmetros utilizados:

Tabela 3. Dados de entrada

Alta Prioridade:	Valores	Baixa Prioridade:	Valores
λ_1	7	λ_2	7,3
pr_1	0,1	pr_2	0
pr_3	0,09	pr_4	0,67
Buffer Médio	5	Buffer Médio	5

Para implementar a política de alocação de recursos é necessário estudar o comportamento da CPU. Assumindo que a tabela acima representa uma situação onde o escalonador esta altamente ocupado e que as entradas possuem um comportamento poissoniano, uma nova aplicação é adicionada em λ_1, simulando uma situação de grande carga de trabalho. A Tabela 4 apresenta a saída do modelo markoviano. Como esperado, quanto maior é a taxa de entrada, maior é a vazão (tabela 4), assim como maior é o tempo em fila e a probabilidade de bloqueio.

Tabela 4. Medidas de Desempenho

Tempo de Espera em Fila							
	0%	10%	20%	30%	40%	50%	60%
S1 / P1	0,48	0,51	0,54	0,56	0,57	0,59	0,60
S2 / P2	4,18	5,46	7,11	9,19	11,80	15,05	19,03
Probabilidade de Bloqueio							
	0%	10%	20%	30%	40%	50%	60%
S1 / P1	0,23	0,27	0,31	0,35	0,39	0,42	0,45
S2 / P2	0,91	0,93	0,94	0,96	0,97	0,97	0,98
Tamanho da fila							
	0%	10%	20%	30%	40%	50%	60%
S1 / P1	2,88	3,13	3,34	3,52	3,67	3,80	3,91
S2 / P2	4,84	4,88	4,91	4,93	4,95	4,96	4,97

Onde 0% representa o comportamento do desempenho do sistemacom apenas λ_1 e λ_2. λ_p é derivado de λ_1 (10%, 20% até 60%) e representa o impacto da adição de uma nova aplicação ao sistema.

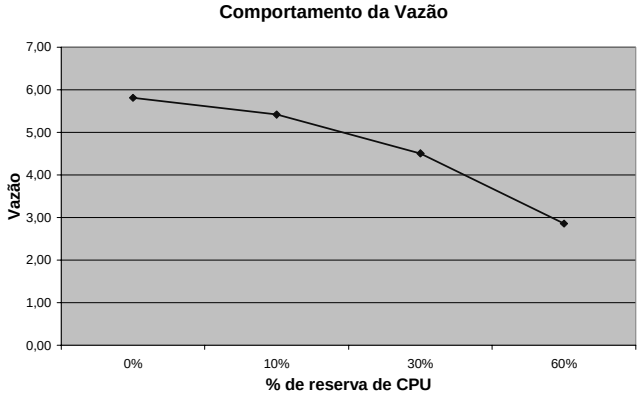

Figura 4. Comportamento da Vazão

O principal objetivo de modelar o processo de escalonamento do kernel é o de avaliar e estudar possíveis modificações objetivando uma melhoria no desempenho das aplicações, no caso da arquitetura Par_QoS, para aplicações paralelas. O modelo ainda esta sendo estudado e testado, entretanto já apresenta alguns resultados interessantes como na Figura 5. Os mesmo testes foram realizados para este modelo, contudo utilizando um processo de escalonamento diferenciado, com reserva de CPU de 60%, o que significa que, 60% da capacidade do processador foi alocada para as aplicações paralelas e os outros 40% para as outras aplicações. Na Figura 5 pode é apresentado o comportamento do sistema de escalonamento normal com a limitação de CPU (100%, 70% e então 40%), e pode ser observado que quanto menor é o percentual de CPU utilizado, menor é a vazão do sistema para ambas as aplicações, entretanto para as aplicações paralelas (parte inferior da figura), quando maior é o percentual de reserva utilizado (0%, 30% e 60%) maior é a vazão da aplicação paralela.

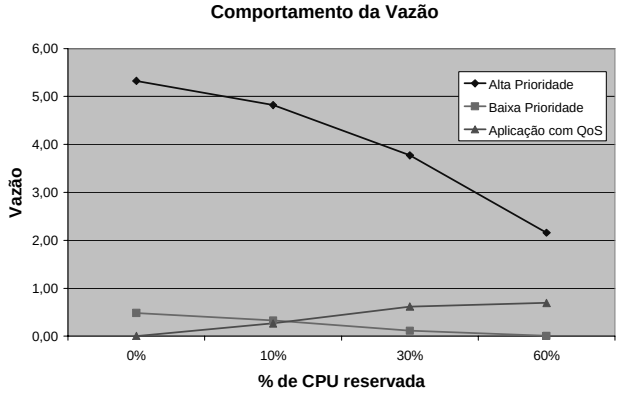

Figura 5. Comportamento da Vazão para política de alocação de recursos

5. Conclusão

Prover garantias de QoS em escalonadores de GPOS é um problema em aberto. Neste artigo é apresentado e proposto um modelo de desempenho para um escalonador markoviano de GPOS e ainda é apresentado um modelo estendido do escalonador que utiliza a política de alocação estática de recursos. Através da análise dos resultados foi concluído que o houve um bom desempenho das aplicações com QoS entretanto, as

outras aplicações sofreram algumas limitações. As contribuições deste artigo são: (a) foi proposto um modelo markoviano para o escalonador do GPOS. Até o momento que este artigo foi escrito, não foi encontrado nenhum outro artigo na literatura que abordasse modelos de desempenho, que é o problema estudado neste artigo; (b) foi proposto e apresentado chamadas de sistema que foram especialmente criadas para mensurar dados importantes utilizados para parametrizar os modelos; (c) foi proposto um modelo markoviano para reserva de recursos que visa ao entendimento do funcionamento do mecanismo de escalonamento do Gnu/Linux. A partir desse entendimento, podem ser realizadas inferências e proposições de alternativas viáveis para a execução de aplicações paralelas com qualidade de serviço.

Atualmente, estão sendo feitos experimentos utilizando técnicas processo markoviano de decisão para encontrar políticas de escalonamento ótimas que objetivem atender os processos paralelos sem inviabilizar a execução dos demais processos.

6. Agradecimentos

Os autores agradecem ao Dr. Solon Carvalho pela utilização do software de modelagem estocástica (MODESTO) [Francês et al. 2005].

Referências

Foster, I.; Kesselman, C.; Tuecke, S. 2001. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” International Journal of Supercomp, Applications, Vol, 15, No,3.

Francês, C., Oliveira, E., Costa, J., Santana, M., Santana, R., Bruschi, S., Vijaykumar, N., Carvalho, S. 2005. “Performance Evaluation based on System Modelling using Statecharts extensions”. Simulation Practice and Theory - Elsevier, Vol. 13, n. 7, 584-618

Kawasaki, R.; Oliveira, L.; Francês, C.; Cardoso, D.; Coutinho, M.; Santana, A. 2003. “Towards the parallel computing based on Quality of Service”, In Proceedings of 2nd IEEE International Symposium on Parallel and Distributed Computing, p, 131-136, Slovenia.

Love, R. “Linux Kernel Development”. 1st edition, SAMS, 2003.

Meo, M.; Marsan, A. j. 2004. “Resource Management Policies in GPRS Systems”, Performance Evaluation, Vol, 56, pp,73-92.

Niyato, D.; Hossain, E.; 2005. “Analysis of Fair scheduler and Connection admission Control in Differentiated Services Wireless Networks”, IEEE International Conference on Vol, 5, 16-20 Page(s):3137 – 3141.

Roy, A.; Foster, I.; Gropp, W.; Karonis, N.; Sander, V.; Toonen, B. 2000. “MPICH-GQ: Quality-of-Service for Message Passing Programs”, Proceedings of the IEEE/ACM SC2000 Conference.

Zhang S, Y.; Moreira, J. 2001. “Impact of Workload and System Parameters on Bext Generation Cluster Scheduling Mechanisms”, IEEE Transactions on Parallel and Distributed Systems, Vol, 12, No, 9, September.