Uma Arquitetura para Provimento de Computação Paralela em Sistemas Operacionais de Propósito Geral

Regiane Y. S. Kawasaki[1], Luiz Affonso H. G. de Oliveira[2], Carlos R. L. Francês[1], Diego L. Cardoso[1], Luiz D. C. Augusto[1], Marcelino S. Silva[1], Mario M. Teixeira[3], João C. W. A. Costa[1]

[1]Laboratório de Computação Aplicada (LACA) – Universidade Federal do Pará (UFPA)
Caixa Postal 479 – 66.075-110 – Belém – PA – Brasil

[2]Departamento de Engenharia de Computação e Automação, Centro de Tecnologia,
Universidade Federal do Rio Grande do Norte (UFRN), 59.072-970,
Natal – RN – Brasil

[3]Departamento de Informática – Universidade Federal do Maranhão (UFMA)
65.085-580 – São Luís – MA – Brasil

{kawasaki,rfrances,jweyl}@ufpa.br, affonso@dca.ufrn.br,
{diego,ldaugusto,marcelino}@deec.ufpa.br, mario@deinf.ufma.br

Abstract. *The increasing demand for Parallel Computing as a means to allow high performance processing has led to the development of several highly specialized computer architectures. This type of architectures, however, has some drawbacks, such as: (1) the high costs involved, and (2) the dedicated nature of these platforms, what usually creates a high degree of idleness. This work presents Par_QoS, an architecture for the provision of differentiated services in a general purpose operating system (in this case, Linux OS), thus allowing seamless parallel computing in traditional, non-dedicated, network environments.*

Resumo. *A crescente demanda pela utilização de Computação Paralela como forma de permitir processamento de alto desempenho vem incentivando a diversificação das arquiteturas de computadores voltadas a essa finalidade. Entretanto, esse tipo de arquitetura possui algumas restrições, tais como: (1) os altos custos envolvidos e (2) o caráter dedicado dessas plataformas (fato que gera, via de regra, um grande grau de ociosidade). Este trabalho apresenta Par_QoS, uma arquitetura para provisão de serviços diferenciados em um sistema operacional de propósito geral, neste caso o Linux, cujo objetivo é permitir uma eficiente computação paralela em ambientes de rede tradicionais (não-dedicados).*

1. Introdução

A computação paralela, nos moldes atuais, pode ser realizada em diversos ambientes. Esses ambientes, via de regra, são escolhidos de acordo com a natureza da aplicação paralela a ser executada. Entretanto, um dos motivos determinantes para a escolha do tipo de arquitetura paralela é o custo [Foster et al. 2002]. Uma alternativa ao alto custo apresentado pelos multiprocessadores vem sendo o uso de *clusters* de computadores pessoais. Mesmo economicamente mais viáveis e com poder de processamento próximo ao das máquinas paralelas, esse tipo de arquitetura ainda é de propósito específico, utilizada de maneira dedicada, gerando, via de regra, ociosidade de recursos, o que na atual conjuntura é extremamente indesejável.

Uma possível alternativa à diminuição da ociosidade, sem expandir o custo final da arquitetura, seria a utilização de redes de computadores não dedicadas, em conjunto com um sistema operacional de propósito geral (GPOS – *General Purpose Operating System*), comumente instaladas na maioria das organizações. Entretanto, apenas usar esses elementos não cria um cenário favorável à execução eficiente de programas paralelos. Isso ocorre devido a várias restrições tecnológicas, principalmente nos GPOSs. Podem ser citadas como restrições existentes [Moreno e Soares 2004]: (1) a política de escalonamento desses sistemas operacionais que destinam quantum para processos ou (2) recorrem apenas ao expediente do uso de prioridades para privilegiar a execução de algumas aplicações em detrimento de outras; (3) seus subsistemas de rede são orientados a interrupções, o que pode causar ineficiência no escalonamento de processos; (4) normalmente, as filas de transmissão de pacotes são compartilhadas, não havendo meios para classificação ou priorização dos pacotes; (5) os protocolos ora utilizados não tratam com naturalidade serviços diferenciados, e (6) os próprios ambientes de passagem de mensagens (como *Parallel Virtual Machine* - PVM e *Message Passing Interface* - MPI) não geram os seus processos com padrões de qualidade de serviço.

Dessa forma, para viabilizar-se computação paralela de forma não dedicada em redes de propósito geral, sem interferências no seu funcionamento ordinário, é necessário implementar a filosofia de tratamento diferenciado de processos paralelos tanto nos sistemas operacionais das máquinas local e destino quanto no canal de comunicação [Roy et al. 2000].

Este artigo apresenta uma proposta de arquitetura, denominada Par_QoS, baseada na inserção do conceito de serviços diferenciados em um GPOS (neste caso o Linux) com o objetivo de permitir computação de alto desempenho em redes tradicionais não dedicadas, ampliando a possibilidade da utilização da Computação Paralela e tornando mais favorável a relação custo/desempenho. A solução proposta pela adoção do Par_QoS pode ampliar o leque de potenciais usuários de programação paralela, o que impõe um caráter mais universal a esse tipo de computação.

Este artigo está organizado da seguinte maneira: a seção 2 mostra os trabalhos relacionados com o apresentado por este artigo. A seção 3 descreve a arquitetura proposta, dando ênfase aos elementos do GPOS que atuam diretamente para provisão de QoS: o subsistema de processamento e o subsistema de rede. A seção 4 apresenta um estudo de viabilidade da arquitetura por meio de prototipação. Por fim, a seção 5 destina-se às considerações finais sobre o trabalho.

2. Trabalhos Correlatos

A literatura especializada apresenta um leque de trabalhos voltados à provisão de serviços diferenciados a tipos diversos de aplicações, tais como as de tempo-real e multimídia, em sistemas operacionais convencionais. Alguns trabalhos propõem a criação de novos sistemas operacionais. Entretanto, a grande maioria se concentra em melhorar as técnicas de escalonamento, seja por meio de pequenas alterações no *kernel* do sistema operacional ou mesmo propor novos métodos de escalonamento a serem inseridos no sistema operacional.

Vale ressaltar que a relação entre o Par_QoS e os trabalhos apresentados a seguir, encontra-se basicamente na proposta de modificações no *kernel* de um sistema operacional, com o objetivo de aperfeiçoar o escalonamento de processos. No Par_QoS essas modificações são basicamente para privilegiar processos paralelos MPI, nos demais, essas mudanças beneficiam processos de aplicações tempo-real e/ou interativa.

O trabalho de [Bruno et al. 1998] propõe a implementação do sistema Eclipse/BSD, baseado no FreeBSD® para permitir que aplicações *soft real-time*, multimídia e web obtenham tratamento diferenciado por parte do sistema operacional. A idéia central do Eclipse/BSD, que é um sistema operacional de tempo-compartilhado, é realizar um compartilhamento de recursos proporcional e hierárquico. Para isso, um arquivo especial chamado */reserv* faz a associação de recursos reservados a objetos compartilhados através de referências.

O escalonador responsável pela CPU utiliza os algoritmos MTR-LS (*Move-To-Real List Scheduling*) para gerenciar o acesso e uso da CPU por parte dos processos. Neste esquema, quando um processo é bloqueado (por exemplo, esperando por alguma operação de E/S), o MTR-LS mantém a porção não utilizada da cota do processo na mesma posição na lista de escalonamento. Desta forma, o processo não é retirado da lista quando se torna pronto para executar novamente. A conseqüência direta do uso dessa estratégia é a diminuição no tempo de execução de processos I/O-bound.

Os algoritmos disponíveis no Eclipse/BSD são: YFQ (*Yet another Fair Queuing*) responsável pelo escalonamento de disco; o WF^2Q (*Worst-case Fair Weighted Fair Queuing*) [Bennett and Zhang 1996] responsável pelo escalonamento de pacotes na interface de saída da rede e o SRP (*Signaled Receiver Processing*) [Brustoloni et al. 2000] responsável pela interface de entrada de pacotes da rede.

O QLinux, proposto por [Sundaram et al. 2000], é um sistema operacional multimídia baseado no GNU/Linux. Ele dá suporte à QoS para aplicações multimídia, *soft-real time* e interativa. O QLinux é formado pelos seguintes elementos: Tanto o escalonador de CPU quanto o escalonador de envio de pacotes à rede utilizado é o *Hierarchical Start Time Fair Queuing* (H-SFQ); usa o subsistema de rede *Lazy Receiver Processing* [Druschel e Banga 1996] para tratar os pacotes que chegam via rede.

Em [Moreno e Soares 2004] é apresentada a arquitetura QoSOS que provê QoS adaptável nos subsistemas de rede e de escalonamento de processos multimídia em GPOS. Em nosso trabalho, também há um arquitetura para provisão de QoS adaptável no Linux. Entretanto, as alterações nos diversos subsistemas da máquina, como por exemplo, escalonamento de processos e rede de comunicação visam promover a execução de aplicações paralelas MPI com QoS.

3. Modelo Arquitetural do Par_QoS

A Figura 1 apresenta uma visão geral da arquitetura Par_QoS [Kawasaki et al. 2003]. Este modelo realiza as seguintes etapas para provisão de QoS em tempo de execução:

(a) A aplicação MPI solicita ao sistema que seus processos recebam tratamento diferenciado em relação aos demais; (b) O sistema operacional recebe as solicitações de serviço dos processos paralelos, como por exemplo, garantias de acesso à CPU e reservas de recursos da rede; (c) O sistema operacional verifica, por meio do módulo controlador de admissão, a viabilidade da aceitação dos fluxos de processos paralelos; (d) Caso o controlador de admissão garanta os recursos aos fluxos de processos paralelos, um contrato de serviço é estabelecido e os fluxos são escalonados à CPU ou ao subsistema de rede; (e) Caso o controlador de admissão não aceite as solicitações da aplicação MPI, o módulo negociador tentará redefinir alguns parâmetros de qualidade para que os processos da aplicação MPI executem satisfatoriamente. Se mesmo assim o negociador não conseguir reservar os recursos a contento, o contrato de serviço não é estabelecido e uma mensagem é enviada ao usuário para que ele tente mais tarde com os mesmos parâmetros ou redefina imediatamente, para baixo, seus parâmetros de qualidade; (f) O sistema operacional tem que garantir a manutenção de serviço, ou seja, manter os valores de QoS ao longo da execução do processo; (g) Realizar as transferências de informações entre processos paralelos, via rede de comunicação com garantias de QoS; (h) Encerrar o contrato de serviço ao término do processo MPI.

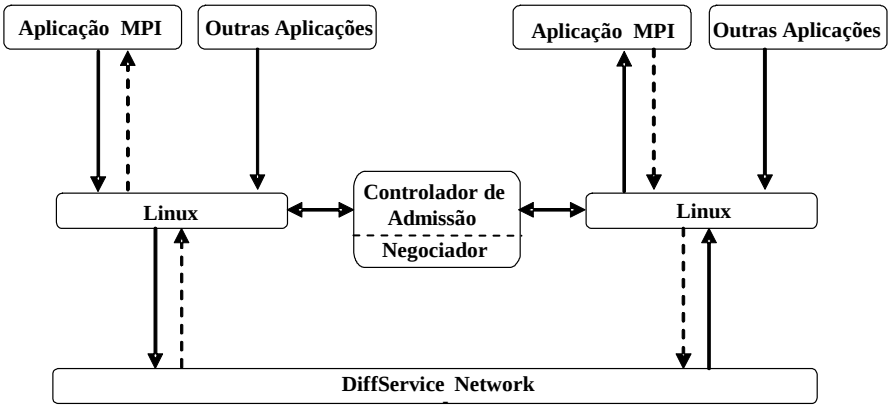

Figura 1: Arquitetura Par_QoS

3.1. Subsistema de Processamento: Escalonamento de Processos

O escalonador tradicional do Linux visa ao equilíbrio entre todos os processos, dessa forma não direciona grandes benefícios ou garantias a processos individuais. Para a implementação da arquitetura Par_QoS, são propostas alterações no escalonador de processos, de tal forma que este seja capaz tratar os processos tradicionais e os processos com QoS com algumas diferenças: enquanto os processos tradicionais seriam escalonados exatamente da forma tradicional do Linux, aqueles processos com QoS seriam escalonados através de verificações no temporizador do sistema.

O Linux (assim como outros GPOS) utiliza o temporizador como forma de realizar rotinas de verificação, como atualização da hora do sistema ou verificação do término do *timeslice* de um processo [Love 2003]. No Linux 2.6[1], a freqüência do temporizador é de 1000 Hz (ou simplesmente uma interrupção a cada milissegundo) – para máquinas da arquitetura i386 (os valores mudam para cada arquitetura).

Para que o Linux reconheça os processos que demandam QoS, foram criadas uma nova estrutura de dados, denominada *task_qos*, e duas novas chamadas de sistema, denominadas *setqospriority()* e *rmvqospriority(),* com as seguintes funções:

- *task_qos*: estrutura de dados em fila circular que referencia um processo que demande reserva de recursos, bem como a quantidade de reserva e próximo momento de entrada no processador.
- *setqospriority(int pid, int reserva)*: insere uma nova entrada na fila dos processos com QoS, instanciando uma task_qos para o processo referenciado.
- *rmvqospriority(int pid)*: retira o processo referenciado da fila com QoS, desalocando a *task_qos* referente a esse processo.

A proposta Par_QoS consiste em adicionar novas verificações referentes aos processos com QoS na rotina de interrupção do temporizador, *scheduler_tick*(): assim a entrada dos processsos com QoS na CPU é forçada. O momento de entrada é representado pela variável *jiffies_to_in* na *task_qos* do processo em questão, e é definido como a porcentagem de reserva solicitada, considerando períodos de *1 segundo*. Após consumir seu *timeslice*, o instante *jiffies_to_in* do processo é definido pelo *complemento* à 1 segundo. Dessa forma, para uma reserva de 20%, o processo terá *timeslice* de 200ms e depois aguardará por 800ms.

Após o término do seu *timeslice*, esses processos com QoS são colocados para esperar na fila junto com outros processos comuns que tiveram seu *timeslice* esgotado (vetor de processos expirados do Linux 2.6), entretanto, com a pior prioridade e o menor *timeslice* definidos pelo SO. Essa atitude possibilita o reescalonamento desse processo com QoS (para processar sem reserva de recursos) antes que o *tick* da próxima entrada no escalonador estoure, revelando uma característica adaptativa da utilização da CPU, mediante a ociosidade da mesma, e tendo como mínimo o próprio valor de reserva.

3.2. Subsistema de Rede

Na arquitetura Par_QoS o subsistema de rede é organizado em dois módulos: (a) o que trata da recepção de dados e (b) o que versa sobre a entrega de dados à rede. O mecanismo adotado para controlar a recepção de dados é o *Lazy Receiver Processing* (LRP), já o mecanismo escolhido para o envio de dados à rede é o algoritmo de escalonamento por prioridade (Figura 2), onde os processos paralelos MPI sempre têm prioridade de envio em relação aos demais processos.

No *Lazy Receiver Processing* (LRP) – Processamento tardio do receptor, proposta por [Druschel and Banga 1996], alguns dos problemas de recepção de dados do subsistema de rede dos GPOSs são resolvidos pela seguinte combinação de técnicas:

[1] Versão do *kernel* utilizada como base de implementação do Par_QoS.

1. Substituição da fila compartilhada IP por filas de *socket*, de acesso direto pela interface de rede. A interface deve, então, demultiplexar os pacotes de entrada colocando-os nas filas apropriadas, de acordo com o *socket* destino.
2. O protocolo de recebimento é executado na prioridade do processo receptor. Para isso, o processamento será executado mais tarde, no contexto da chamada de sistema disparada pelo processo responsável pelo recebimento.

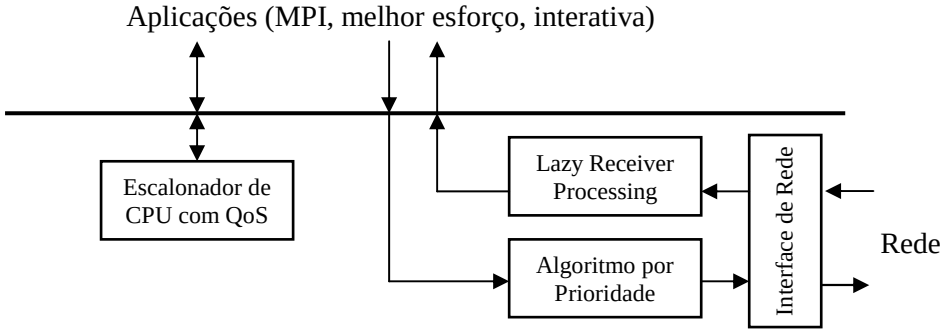

Figura 2: Subsistema de rede do Par_QoS

Assim, o processamento do protocolo para um pacote não ocorre até que a aplicação receptora o requisite pela chamada de sistema. Além disso, esse processamento não mais interrompe o processo em execução no momento da chegada do pacote, a não ser que o receptor tenha maior prioridade no escalonamento que o processo corrente. Evitam-se, ainda, mudanças de contexto inapropriadas, levando a um significativo aumento do desempenho do sistema.

4. Avaliação de Desempenho do Par_QoS

Para comprovar a viabilidade da proposta do Par_QoS, foi implementado um *testbed* no qual são executadas diversas aplicações com diferentes prioridades e percentuais de reserva de recursos. Os resultados obtidos são apresentados e comparados a seguir.

- **Descrição do Ambiente Operacional e Softwares Empregados:**

Em termos de hardware, o protótipo foi avaliado no Laboratório de Computação Aplicada (LACA) da Universidade Federal do Pará, no qual existem 13 computadores, entre estações e servidores. Objetivando avaliar o desempenho do sistema operacional, as máquinas envolvidas foram interconectados por uma rede local *Ethernet*, chaveada por um SWITCH 3COM SUPERSTACK II 1000, cujo suporte de taxa de transferência é de até 155Mbps, desta forma gerando atrasos mínimos na rede. As estações utilizadas foram 1 Pentium 4 2.6GHz (como máquina *mestre*) e 2 *Athlon* 2200 (como *escravos*), todos com 256 MB de RAM DDR 333 e placas de rede *PCI Fast Ethernet* 10/100.

Quanto ao software básico, todas as estações utilizadas na avaliação utilizam a distribuição *Linux Fedora Core 3*, bem como o ambiente de passagem de mensagens LAM-MPI 7.1.1. Para medição do desempenho de aplicações seqüenciais e paralelas, utilizou-se o como *benchmark* um programa de cálculo que utiliza o *Quantum Monte Carlo* [QMcBeaver 2004], já utilizado como *benchmark* por empresas de desenvolvimento de compiladores [Pathscale 2004].

Para demonstração da validade da arquitetura mesmo em ambientes de produção altamente carregados, utilizaram-se três aplicações seqüenciais de multiplicação de matrizes de alta ordem como geradores de carga nas estações *escravas*. Estas aplicações naturalmente ocupam a CPU quase que totalmente, caracterizando-se como aplicações *CPU-bound*, um atributo pertinente ao teste a ser realizado.

- **Análise dos resultados obtidos:**

O primeiro experimento consiste na execução dos três programas seqüenciais nas estações *escravas* conjuntamente com *benchmark* paralelo em diferentes situações: quando sobre reserva de CPU de 40%, reserva de CPU de 70% e quando não há nenhum tipo de reserva de processamento. O experimento foi repetido, no mínimo, trinta (30) vezes para composição de uma amostra de uma distribuição Gaussiana.

Tabela 1. Média dos tempos de resposta (em segundos) do experimento, e seus Speedups e Eficiência [Foster et al. 2002], para quatro situações: seqüencial, paralelo sem reserva, paralelo com reserva de 40% e com reserva de 70%

	Seqüencial	Paralelo sem Reserva	Paralelo com 40%	Paralelo com 70%
Duração	2023,170	602,359	336,720	232,343
SpeedUp	1	3,358	6,008	8,707
Eficiência	1	1,119	2,002	2,902

Para os experimentos realizados (ver Tabela 1), o desempenho da arquitetura Par_QoS foi bastante satisfatório para as situações com reserva de recursos. Notoriamente, a situação de reserva de 40% de recursos, na qual se obteve um tempo da ordem da sexta parte (1/6) do tempo seqüencial e cerca de 55% do valor obtido com o programa paralelo sem reserva. Assim, em relação ao tempo da aplicação paralela sem reserva, houve uma redução de tempo de 45% e 61,5% para as situações de reserva de 40% e 70%, respectivamente.

Pela grande flexibilidade de configuração dos percentuais de reserva, o Par_QoS pode facilmente ser adaptado às especificidades de cada conjunto de aplicações, de maneira a não priorizar demais ou de menos alguma classe de processos.

Um ponto importante na utilização de Par_QoS é a garantia de que as demais aplicações que estão coexistindo na rede não sejam penalizadas (o que poderia levar a uma situação de inanição). Para garantia de execução satisfatória dos demais processos do sistema, durante o experimento, foi observada uma medida de avaliação de programas seqüenciais, executados concorrentemente com programa paralelo, a fim de observar o grau de atraso inserido nas aplicações.

Como medida definida, foi utilizado o tempo de espera por processamento na fila de processos, e como aplicação seqüencial foi utilizada a execução de um arquivo de vídeo comprimido (não *stream*) no mesmo ambiente computacional anteriormente citado. A aplicação foi escolhida por sua alta interatividade e grande utilização de largura de banda. Dessa forma, um aumento significativo no tempo do processo na fila reduziria em demasia a qualidade da aplicação. O experimento observou também o tempo de espera na fila para as aplicações de multiplicações de matrizes, utilizadas como geradoras de carga.

Tabela 2. Tempo médio (em milissegundos) na fila para a aplicação de vídeo e de matrizes nas diferentes situações de reserva do programa paralelo

	Disputa sem reserva	Disputa com reserva de 40%	Disputa com reserva de 70%
Vídeo	328,308	471,486	1092,756
Matrizes	101,677	132,502	256,852

Observa-se na Tabela 2 que, como desejado, o tempo de fila para execução do vídeo aumentou progressivamente, muito embora esse aumento não tenha causado impacto significativo na reprodução do vídeo, já que o mesmo continuou executando sem paradas ou deterioração na qualidade do som, sendo notado apenas a perda de alguns *frames*.

Entretanto, é importante observar que a relação tempo de fila e aumento de reserva de recursos não é linear, ou seja, o tempo de espera em fila não cresce na mesma proporção do crescimento da reserva de recursos. Assim, para a passagem do seqüencial para a reserva de 40%, houve cerca de 40% de aumento no tempo de fila para a aplicação de vídeo e cerca de 30% para a aplicação de multiplicação de matrizes. Já na passagem do seqüencial para a reserva de 70% de recursos, o tempo de espera em fila cresceu cerca de 330% para o vídeo e 250% para a aplicação de multiplicação de matrizes.

Quando a comparação é feita entre as situações de reserva de recursos, também há uma relação não linear. Assim, com o aumento de 40% para 70% de reserva, ou seja, um acréscimo de 75% na reserva, o tempo de fila aumentou cerca de 250% para a aplicação de vídeo e 200% para a aplicação de multiplicação de matrizes.

Essa análise leva à interpretação de que para cada conjunto de aplicações há um valor ótimo para reserva de recursos, sendo que esse valor é função direta do conjunto das aplicações. Entretanto, pela grande facilidade de reconfiguração dos níveis de reserva dentro do Par_QoS, essa adaptação é facilmente realizada de acordo com as necessidades do conjunto de aplicações.

Para o experimento, também foi realizada uma análise sob a óptica da taxa de utilização do processador. Essa análise da taxa de utilização da CPU pelas aplicações seqüenciais de multiplicação de matrizes compara as diferentes situações de reserva com o programa paralelo. De acordo com o gráfico da Figura 3, nota-se que quando competem somente entre si, as aplicações de multiplicação ocupam cerca de 30% da CPU, cada uma, e cerca de 25% quando competindo entre si e com o programa paralelo sem qualquer tipo de reserva. Essas duas situações demonstram uma divisão igualitária da utilização da CPU, proporcionada pelo escalonador de processos do Linux. Entretanto, quando sob o domínio da arquitetura Par_QoS, as taxas de utilização para os programas de multiplicação de matrizes caem para 17% e 9% nas situações de reserva de 40 e 70%, o que denota uma divisão igualitária entre as aplicações seqüenciais, entretanto somente dentro da porcentagem da CPU não reservada.

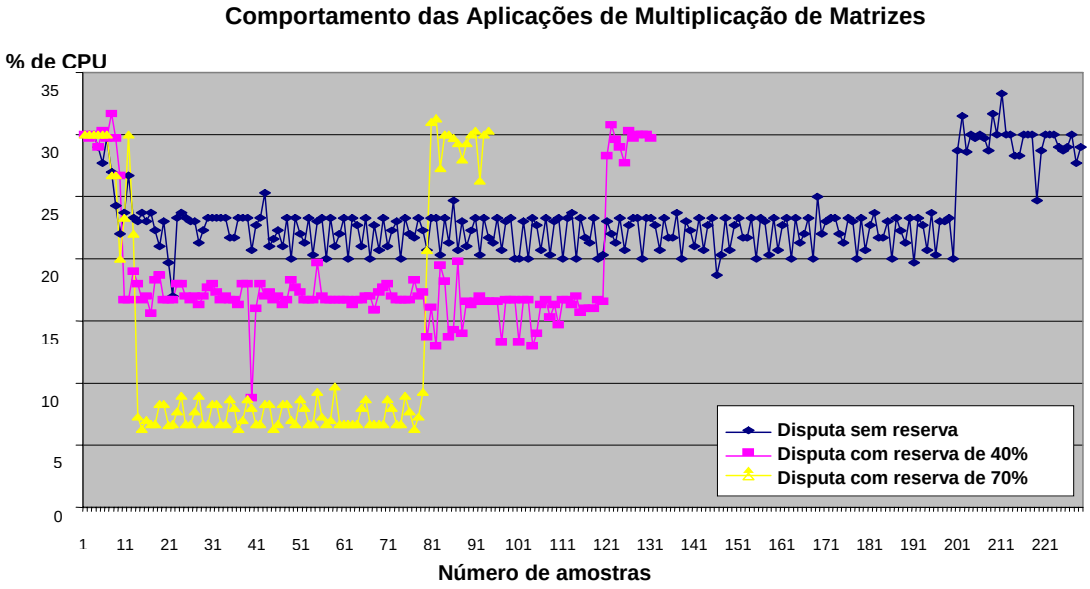

Figura 3. Taxas de utilização da CPU por aplicações CPU-bound

O gráfico da Figura 4 apresenta a evolução dos tempos de espera em fila (Y) para a aplicação de vídeo em função do conjunto de 259 execuções (X).

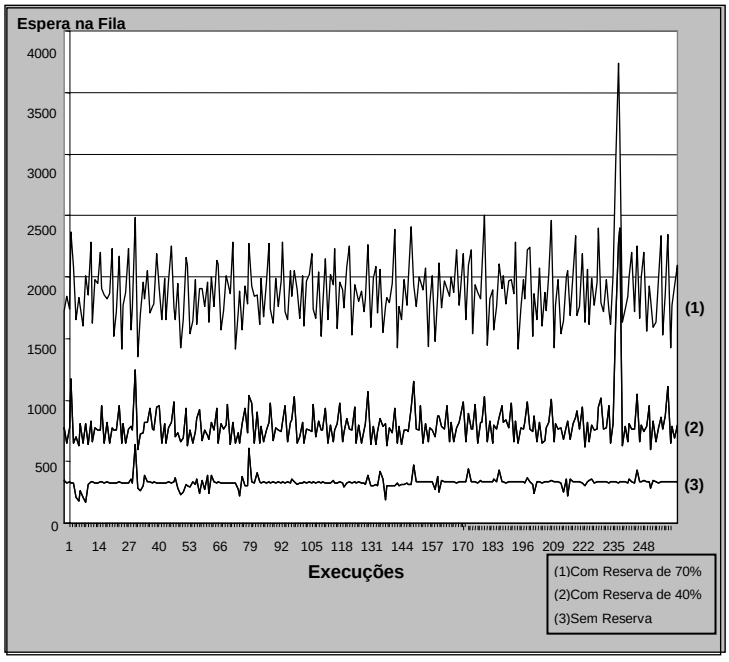

Figura 4. Comportamento da aplicação de vídeo para as situações: sem reserva de recursos, com reserva de 40% e com reserva de 70%

6. Considerações Finais

Este artigo apresenta uma proposição de arquitetura para prover computação paralela sobre um sistema operacional de propósito geral. Há uma série de benefícios em adotar-

se uma estratégia com o perfil apresentado neste trabalho. A proposta traz algumas contribuições, destacando-se: (a) viabilização de utilização de aplicações de computação paralela em redes não dedicadas e com sistemas operacionais de propósito geral; (b) facilidade de configuração e reconfiguração do ambiente e dos percentuais de reservas, de acordo com o perfil da aplicação; (c) utilização de padrões de direito ou de fato ao longo da arquitetura (MPI, TCP/IP e Linux); (d) diminuição dos custos de implementação da plataforma computacional, pela utilização de ferramentas gratuitas e código aberto e infra-estrutura de rede já existente; (e) a reserva de recursos através do acréscimo de 1 linha de código na programação; (f) ferramenta que pode ser utilizada, de forma didática, no ensino das disciplinas sistemas operacionais, computação paralela e sistemas distribuídos.

Atualmente, estão sendo definidas classes de aplicações, através de estudos de redes *bayesianas*, investigando-se parâmetros da aplicação, tais como: utilização de processador e memória, largura de banda necessária, grau de interatividade e tempo de resposta.

Referências Bibliográficas

Bennett, J.C.; Zhang, H. (1996). "WF2Q: Worst-Case Fair Weighted Fair Queuing". In Proceedings of IEEE INFOCOM'96, San Francisco, CA, USA, p.120-128.

Bruno, J.; Gabber, E.; Özden, B.; Silberschatz, A. (1998). "The Eclipse Operating System: Providing Quality of Service via Reservation Domains". In Proceedings of USENIX Annual Technical Conference, New Orleans, Louisiana, USA.

Brustoloni, J.; Gabber, E.; Silberschatz, A. (2000). "Signaled Receiver Processing". In Proceedings of the USENIX Annual Technical Conference, San Diego, CA, p. 211-223.

Druschel, P.; Banga, G. (1996). "Lazy Receiver Processing (LRP): A Network Subsystem". In Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation (ODSI), Seattle, WA.

Foster, I.; Kennedy, K.; Dongarra, J.; Fox, G. (2002). "Sourcebook of Parallel Computing". Morgan Kaufmann Publ.

Kanevsky, A.; Skjellum, A.; Rounbehler, A. (1998). "MPI/RT - An Emerging Standard for High-Performance Real-Time Systems". In Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS31).

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 (ISPDC), p. 131-136, Slovenia.

Love, R. (2003). "Linux Kernel Development". 1st edition, SAMS.

Moreno, M.; Soares, L.F. (2004). "QoSOSLinux: Desenvolvimento de uma extensão Linux com suporte adequado a qualidade de serviço". I WSO - Workshop de Sistemas Operacionais, XXIV Congresso da Sociedade Brasileira de Computação – SBC, Salvador, Brasil.

QMcBeaver (2004) "QmcBeaver Project". Dsiponível em http://sourceforge.net/projects/qmcbeaver.

Pathscale (2004). "Pathscale Compiler Suite". http://www.pathscale.com/performance.html.

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

Sundaram, V.; Chandra, A.; Goyal, P.; Shenoy, P.; Sahni, J.; Vin, H. (2000). "Application Performance in the QLinux Multimedia Operating System". ACM Multimedia, Los Angeles, CA.