Avaliação e Otimização do Novo Escalonador do Linux para Ambientes Distintos

Tiago J. S. Lopes[1], Davi P. Santos[1],
José Arnaldo M. de Holanda[1], João M. Arcoverde[1],
Nils M. Larsgard[2]

[1]Departamento de Ciência da Computação ICMC - USP
Caixa Posta 668 – 13560-970 – São Carlos – SP.

[2]Norwegian University of Science and Technology (NTNU)
NO 7491 Trondheim – Norway

{tiago, davi, arnaldo, jmaa}@icmc.usp.br, nilsmagnus@gmail.com

Abstract. *The new Linux kernel version 2.6 seeks the balance between applications of high-processing and applications of high-interactivity, mainly by its new process scheduler, with time complexity of O(1). In the present study we optimized this process scheduler to benefit distinct environments, that is, applications of high-processing, with no response time as a critical necessity, desktop environments, where the response time is a decisive factor in its quality and user satisfaction and also server environments, which has this two characteristics: high processing and many IO operations.*

Resumo. *O novo kernel do Linux versão 2.6 busca o equilibrio entre as aplicações de alto processamento e as de alta interatividade, principalmente através de seu escalonador de processos, com complexidade de tempo constante da ordem de O(1). Neste estudo nós realizamos a otimização deste escalonador para beneficiar ambientes distintos, isto é, ambientes de alto processamento, que não requerem tempos de resposta críticos, ambientes desktop, cujo tempo de resposta é um fator decisivo na determinação de sua qualidade e satisfação do usuário e também ambientes do tipo sevidor, que possuem as duas características: alto processamento e várias operações de entrada e saída.*

1. Introdução

O escalonador de processos é uma parte do kernel responsável por decidir qual processo deverá ser executado pela CPU em um certo momento [Tannenbaum 1992]. Essa escolha é baseada em um algoritmo de escalonamento, que varia de acordo com o objetivo do sistema. Por exemplo, o escalonador pode dar prioridade maior aos processos executados por determinado usuário, visando oferecer um melhor serviço ao mesmo. Pode ainda, priorizar processos que beneficiem uma aplicação crítica que necessita de tempo de resposta pequeno.

Originalmente o sistema operacional Linux ficou conhecido por sua grande utilização em servidores. Contudo, o esforço de usuários e desenvolvedores fez com que o Linux também pudesse ser utilizado em ambientes desktop, tornando-se uma opção neste mercado dominado por tecnologias proprietárias. Um dos grandes passos para tornar o Linux apropriado para ambientes servidores e desktops foi dado por Ingo Molnar

e está presente na versão 2.6 do kernel. O escalonador de processos deste kernel foi desenvolvido para buscar o equilibrio entre os dois ambientes e isto o torna um interessante caso de estudo.

2. Objetivo

Diferentes tipos de usuário possuem necessidades distintas em um sistema operacional. Usuários de desktop desejam que seu sistema beneficie os processos que têm maior interatividade, oferecendo-lhes o melhor tempo de resposta possível. Administradores e gerentes de servidores querem que o sistema operacional execute o maior número de tarefas no menor tempo possível. Cientistas que precisam executar processos de alto processamento, que envolvem muitos cálculos, não necessitam de uma resposta imediata, contudo desejam que execução destas tarefas dure um número mínimo de horas.

O objetivo deste estudo é realizar a avaliação do novo escalonador de processos presente no kernel do Linux versão 2.6, comparando-o ao kernel 2.4.26 e otimizando-o a fim de melhor adaptar-se a aplicações interativas *(IO-bound)*, aplicações computacionalmente intensivas *(CPU-bound)* e aplicações de servidor, que como mencionado acima, representam necessidades distintas e conflitantes.

3. Escalonador do Kernel 2.6

Uma das mudanças mais significativas que ocorreu durante o desenvolvimento do Linux 2.5 foi o seu novo algoritmo de escalonamento de processos [Open Source Development Labs 2003]. Apesar do escalonador do kernel 2.4 ser amplamente utilizado e confiável, ele ainda apresentava algumas características indesejáveis, que estavam fortemente ligadas a seu projeto, o que motivou Ingo Molnar a tentar consertá-lo. Como resultado, foi produzido um escalonador inteiramente novo, ao invés de modificações no escalonador anterior.

Um dos grandes defeitos do algoritmo de escalonamento de processos da versão 2.4 do kernel era a complexidade de tempo do algoritmo, da ordem de O(n), onde n é o número de processos existentes no sistema. Assim, a característica mais festejada do novo escalonador é sua complexidade de tempo constante, da ordem de O(1). Isto significa que o novo algoritmo sempre realiza sua tarefa no mesmo intervalo de tempo, independentemente do número de processos existentes no sistema. Essa característica permite ao kernel do Linux manipular um grande e crescente número de processos sem que o desempenho seja prejudicada de forma linear, como ocorria com o escalonador de processos do kernel 2.4.

No novo algoritmo do escalonador de processos, alguns parâmetros são extremamente importantes para alcançar o equilibrio entre aplicações de alto processamento e de alta interatividade:

- *MIN_TIMESLICE e MAX_TIMESLICE*: representam o valor mínimo e o valor máximo, respectivamente, de timeslice que um processo pode receber. A alteração desses parâmetros para valores maiores permitem a processos permanecerem mais tempo sendo executados pela CPU, diminuindo o número de mudanças de contexto e desta forma beneficiando aplicações CPU Bound.

- *PRIO_BONUS_RATIO*: representa a porcentagem da variação da prioridades dinâmica que um processo pode ter como bônus ou como punição, por sua natureza IO Bound ou CPU Bound. Os processos CPU Bound recebem punições em sua prioridade para que possam ser removidos do estado de execução no momento em que um processo IO Bound já estiver pronto para ser executado. Quanto maiores estes valores, menores serão as prioridades dinâmicas dos processos CPU Bound, e estas serão sempre interrompidas pelos processos IO Bound.
- *MAX_SLEEP_AVG*: quanto maior for o valor desse parâmetro, maior será o tempo que os processos deverão permanecer em estado de espera para serem considerados ativos e voltarem a ser executados. Assim, o aumento desse valor tende a piorar a interatividade.
- *STARVATION_LIMIT*: processos interativos podem causar inanição (starvation) em outros processos, pois eles são executados a qualquer momento, colocando um processo de menor prioridade que está sendo executado no estado de espera (preemption). Assim, se algum processo permanecer em estado de espera por um tempo maior do que este parâmetro, os processos interativos param de ser executados para que esse processo "entre" na CPU.

4. Metodologia

Foram construídas aplicações sintéticas para simular de maneira extrema o comportamento desejado para as aplicações, de forma que a aplicação CPU Bound realizava cálculos de forma ininterrupta, sem operações de entrada e saída. A aplicação do tipo IO Bound realizava de forma intensa operações que envolviam recursos físicos do sistema, como placa de rede e de vídeo. Nesta aplicação, o processamento pode ser minimizado ao máximo, mas não eliminado, pois existe uma espécie de processamento "basal" neste tipo de aplicação. Para a aplicação do tipo servidor, foi utilizada um dicionário feito em PHP que traduz e retorna na tela centenas de palavras, fazendo uma consulta ao banco de dados MySQL para cada uma delas, o código fonte pode ser solicitado ao autor e o programa está disponível para uso em: *http://davi.digitalserver.com.br/dicionario*.

Assim, nos experimentos conduzidos, foram executados os três tipos de processo paralelamente, pois se cada um tivesse sido executado separadamente, não haveria ganho significativo, já que não existiriam outros processos concorrendo por recursos da máquina. Os parâmetros do escalonador foram ajustados com base na teoria, estudando cuidadosamente o artigo de Josh Aas [Aas 2005] e o Kerneltrap [Love 2002]. Foram feitos ajustes finos com a finalidade de encontrar o ponto de estabilidade do sistema e o melhor desempenho para cada caso. Para essa manipulação dos parâmetros foi desenvolvido o sistema de testes que realizava de forma automática as tarefas: 1) rodar os experimentos medindo seus tempos individuais de execução; 2) modificar os parâmetros do escalonador de processos; 3) compilar o kernel e reiniciar a máquina com o novo kernel e voltar ao primeiro passo.

5. Resultados

Os experimentos foram executados 3 vezes com cada conjunto de parâmetros e a média destas execuções foi obtida. Este número de execuções foi escolhido pois em testes preliminares foi constatado que não havia variação significativa entre as execuções, e desta

forma consideramos este valor apropriado e confiável para os experimentos efetivos. Assim, nos gráficos comparativos foi representada a média das três execuções dos programas. O tempo de execução dos programas foi definido em cerca de 30 minutos, sendo que para controlar este tempo de execução, nós manipulamos o tamanho do conjunto de dados que os programas teriam acesso. Notamos também que o término de um dos aplicativos antes dos demais apenas comprova que o objetivo foi alcançado, ou seja, através da modificação dos parâmetros do kernel, foi possível privilegiar um dos diferentes tipos de programas executados.

Primeiramente foi realizado um experimento que comparou a execução dos programas do tipo CPU Bound, IO Bound e do tipo servidor no kernel 2.4.26 e no kernel 2.6.11.11, ambos sem nenhuma modificação, apenas compilados depois de baixados do repositório público do kernel do Linux (http://www.kernel.org). Na figura 1, podemos ver que os tempos de execução para o processo de natureza CPU Bound e do tipo servidor os tempos são similares, já na execução dos processo do tipo IO Bound vemos um desempenho melhor do kernel 2.4.26. Isto pode ser relacionado a um melhor gerenciamento dos recursos da máquina por parte do kernel vencedor neste quesito, e também a possibilidade de existirem minúcias de programação nos algoritmos de escalonamento que fazem com que mesmo a complexidade da ordem de O(n) seja melhor que a complexidade O(1) para um pequeno número de tarefas executadas. Isto deve-se ao fato de normalmente essas nuances nos algoritmos se transformarem em constantes no cálculo de complexidade que normalmente são deixadas de lado na apresentação final das ordens de complexidade [Cormen et al. 1991].

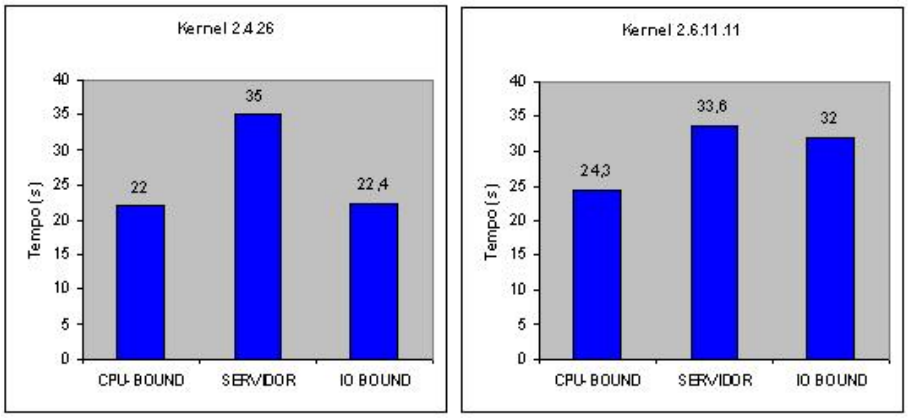

Figura 1. Comparação de desempenho entre duas versões do kernel.

Para favorecer os aplicativos do tipo IO Bound em detrimento dos demais, os melhores valores para os parâmtros mencionados na seção 3. foram:
MIN_TIMESLICE = 5, *MAX_TIMESLICE* = 5, *PRIO_BONUS_RATIO* = 100, *MAX_SLEEP_AVG* = 5 e *STARVATION_LIMIT* = 5000.

Assim, como podemos ver na figura 2, o objetivo foi atingido e os programas do tipo IO Bound foram bastante favorecidos. Além disso, vale notar que os programas do tipo CPU Bound não sofreram prejuízo em seu desempenho. Isto se deve ao ajuste preciso dos parâmetros do escalonador, que de forma inesperada melhoram o desempenho de um tipo de aplicação sem prejuízo das demais.

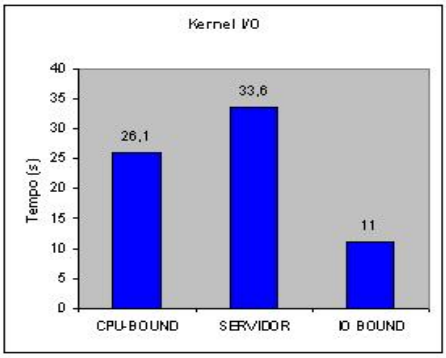

Figura 2. Escalonador modificado para favorecer aplicações IO Bound

Para modificar os valores dos parâmtros de forma a favorecer os processos de processamento intensivo (CPU Bound), os melhores valores obtidos para os parâmetros foram: *MIN_TIMESLICE* = 800, *MAX_TIMESLICE* = 800, *PRIO_BONUS_RATIO* = 3, *MAX_SLEEP_AVG* =800 e *STARVATION_LIMIT* = 1.

Nota-se pela figura 3 que como esperado o programa de natureza CPU Bound foi bem favorecido em detrimento baixo do aplicativo IO Bound, e o processo do tipo servidor também teve um desempenho melhor que na sua execução no kernel com escalonador modificado para favorecer aplicações IO Bound. Com isso verificamos que a natureza do programa servidor, apesar de mista, possui uma parte intensiva de cálculos e comparações vetoriais, ou seja processamento intesivo, benefiado pelas mudanças de otimização para este tipo de aplicação.

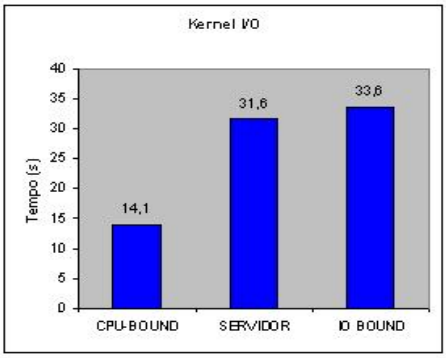

Figura 3. Escalonador modificado para favorecer aplicações CPU Bound

E por fim, a otimização do processo do tipo servidor se revelou uma tarefa difícil, devido a sua natureza híbrida, isto é, possui tanto procesamento intenso quanto operações de entrada e saída. Os valores considerados ótimos para os parâmetros foram definidos da seguinte forma: *MIN_TIMESLICE* = 50, *MAX_TIMESLICE* = 100, *PRIO_BONUS_RATIO* = 34, *MAX_SLEEP_AVG* = 1000 e *STARVATION_LIMIT* = 6.

Os tempo de execução de cada tipo de processo com os parâmetros modificados estão na figura 4.

Foi notado que a natureza do nosso programa servidor era de processamento intenso, desta forma, podemos notar que com o ajuste dos parâmetros para este caso, o

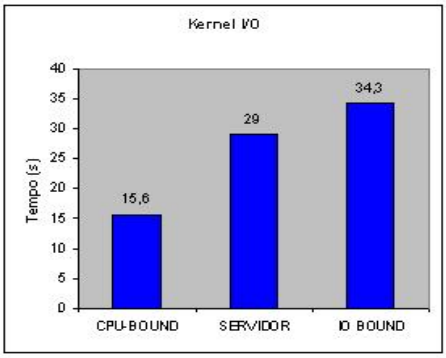

Figura 4. Escalonador modificado para favorecer aplicações do tipo servidor

processo do tipo CPU Bound também foi favorecido, e vale notar também que o processo do tipo servidor não teve ganhos tão significativos como nos outros dois casos, pois é impossível favorecer uma parte do processo servidor (seu processamento), sem sofrer as consequências da perda de desempenho de suas operações de entrada e saída.

6. Conclusões

Neste estudo foi apresentada uma comparação entre o desempenho de aplicações IO Bound, CPU Bound e híbridas (servidores), utilizando o kernel 2.4 e 2.6 do Linux. Também foram realizadas mudanças nos valores do escalonador de processos do kernel versão 2.6.11.11 do Linux que otimizaram o sistema operacional tanto para ambientes de processamento intenso quanto para ambientes interativos, caracterizados por muitas operações de entrada e saída e a necessidade de tempos de resposta aceitáveis. Foi verificado que a otimização de processos híbridos, os quais além de processamento intenso também realizam muitas operações de entrada e saída, é uma atividade não-trivial, mas a observação de seu comportamento, ou seja, se possui mais características IO Bound ou CPU Bound, pode direcionar a otimização do escalonador de processos através da modificação de seus parâmetros reguladores.

Agradecimentos

Agradecemos a Profa. Regina C. Santana pelas contribuições fornecidas ao longo do desenvolvimento deste estudo e a Ronaldo C. Prati pela ajuda com o LaTeX, e aos revisores anônimos que colaboraram com sugestões de melhorias.

Referências

Aas, J. (2005). Silicon Graphics, Inc. (SGI).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (1991). *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2 edition.

Love, R. (2002). Linux: Scheduler tunables. http://kerneltrap.org/node/525.

Open Source Development Labs, I. (2003). Linux process scheduler improvements. http://developer.osdl.org/craiger/hackbench/.

Tannenbaum, A. S. (1992). *Modern Operating Systems*. Prentice Hall, 2nd edition.