Um Esquema para Distribuição de Carga em Ambientes Virtuais de Computação Maciçamente Paralela

Fabiano de O. Lucchese, Francisco S. Sambatti*, Eduardo J. Huerta Yero† e Marco A. A. Henriques
DCA - FEEC - UNICAMP
[flucches,sambatti,huerta,marco]@dca.fee.unicamp.br

Resumo

O recente desenvolvimento de infra-estruturas de telecomunicação, como as redes internacionais de computadores, capazes de interconectar milhões de computadores espalhados pelo mundo inteiro, tornou possível a utilização de extensos recursos computacionais a custos relativamente baixos. Desta nova realidade, surgiram as pesquisas de computadores virtuais maciçamente paralelos, que consistem em ambientes virtuais formados por um grande número de computadores que procuram trabalhar cooperativamente na busca de soluções para problemas até então impossíveis de serem tratados pelos sistemas computacionais disponíveis. Entretanto, como em todo novo domínio de pesquisa, existem também muitas questões ainda sem solução, especialmente em relação ao problema de gerenciamento da carga de processamento realizado nestes ambientes. Neste trabalho, será abordado o problema de balanceamento de carga em ambientes virtuais de computação maciçamente paralela introduzindo-se o algoritmo de Escalonamento Geracional com Replicação de Tarefas (GSTR).

1 Introdução

A integração, via redes digitais, de computadores nos mais variados setores da sociedade e a integração destes setores em uma rede mundial como a Internet, permitiu que um grande número de dispositivos dotados de poder computacional e de armazenamento de dados pudesse interagir de maneira a resolver problemas complexos até então impossíveis de serem resolvidos por um, ou poucos, destes dispositivos isoladamente. Com redes de interconexão cada vez mais rápidas e que conectam um número crescente de computadores, as possibilidades tornaram-se enormes.

Entretanto, são também inúmeras as dificuldades em se tirar proveito deste tipo de recurso; provavelmente as maiores dificuldades residam na impossibilidade de se avaliar com precisão as características de uma rede de computadores como a Internet, em que os conjuntos de computadores e canais de comunicação são extremamente heterogêneos e variam segundo uma dinâmica difícil de modelar.

Neste trabalho é abordada a questão da distribuição de carga computacional em um sistema distribuido tão heterogêneo e passível de falhas quanto se pode esperar de um sistema baseado na Internet. Para isso, é apresentada na seção a seguir algumas definições preliminares sobre as quais se baseará todo o desenvolvimento teórico deste trabalho.

2 Definições preliminares

Apesar das promissoras perspectivas reservadas aos sistemas computacionais maciçamente distribuidos, poucas foram as iniciativas em se formalizar o modelo de computação promovido por este tipo de plataforma [10]. Algumas das iniciativas de formalização de modelos de computação paralela mais bem estabelecidas, tais como [8] e [2], referem-se a sistemas com características diferentes daquelas presentes nos computadores virtuais maciçamente paralelos, sobretudo no que tange à sua escalabilidade. Por esta razão, nesta seção será apresentada uma proposta de modelo de computação adequado às características das plataformas que procuram desenvolver o tipo de processamento descrito anteriormente.

Os conceitos-chave a serem utilizados neste trabalho são os de **aplicação** e de **MPVC**. Definida informalmente, uma aplicação pode ser entendida como um trabalho a ser desenvolvido sobre um conjunto de dados com o objetivo de se produzir algum resultado esperado. Já um MPVC, definido também de modo informal, é a estrutura computacional necessária para se realizar o trabalho representado por uma aplicação.

Apesar de aparentemente simples, estes conceitos, da maneira informal como foram definidos, possuem pontos

*Professor da Universidade Estadual do Oeste do Paraná
†Professor da Univrsidade de Havana

ambíguos e, por isso, devem ser definidos segundo critérios mais rigorosos. Nas sub-seções a seguir serão apresentadas formalizações para estes conceitos, as quais serão utilizadas ao longo deste trabalho.

2.1 O modelo de aplicação

Definição 1 *Define-se um* **bloco de dados** *como sendo uma estrutura capaz de armazenar conjuntos arbitrariamente grandes de dados serializáveis.*

Definição 2 *Define-se uma* **tarefa** τ *como sendo uma seqüência ordenada de operações matemáticas, de desvio condicional ou de atribuição/entrada/saída de dados, que implementa algum tipo de processamento determinístico e finito sobre um bloco de dados de entrada a fim de produzir um bloco de dados de saída.*

Definição 3 *Define-se um* **lote de tarefas** β^m *de multiplicidade* m*, onde* $m \in \aleph^*$*, como sendo a aplicação de uma determinada tarefa sobre* m *blocos de dados de entrada, potencialmente distintos, com o objetivo de se produzir* m *blocos de dados de saída.*

Definição 4 *Se* $\beta_x^{m_x}$ *e* $\beta_y^{m_y}$ *são dois lotes de tarefas distintos e se um sub-conjunto não vazio dos blocos de dados de entrada de* $\beta_y^{m_y}$ *é formado por um sub-conjunto não vazio dos blocos de dados de saída de* $\beta_x^{m_x}$*, então diz-se que há uma* **relação de dependência de dados** *entre* $\beta_x^{m_x}$ *e* $\beta_y^{m_y}$ *e essa relação é denotada por* δ_{xy} *Nessas circunstâncias,* m_y *deve ser um múltiplo ou um sub-múltiplo de* m_x*.*

Definição 5 *Define-se uma aplicação paralela como sendo formada por:*

- *um conjunto* $B = \{\beta_0^{m_0}, \beta_1^{m_1}, \ldots, \beta_N^{m_N}\}, N \geq 0$*, de lotes de tarefas,*
- *um conjunto* $D = \{\delta_{x_0y_0}, \delta_{x_1y_1}, \ldots, \delta_{x_Wy_W}\}$*, de relações de dependência de dados entre lotes de tarefas de* B*, em que as seguintes restrições sejam obedecidas:*

  1. $\forall \delta_{xy} \in D, y > x.$
  2. $\forall \beta_x^{m_x} \in B, x < N, \exists \delta_{ij} \in D \mid i = x.$
  3. $\forall \beta_x^{m_x} \in B, x > 0, \exists \delta_{ij} \in D \mid j = x.$

2.2 O modelo de MPVC

Definição 6 *Define-se uma unidade de processamento* v *como sendo uma entidade capaz de executar qualquer tarefa sobre qualquer bloco de dados a ela submetido, e capaz de se comunicar com outras unidades de processamento. Suas características são definidas pelos conjuntos* V_p *e* V_c*.*

- *Dado um conjunto* $T = \{\tau_0, \tau_1, \ldots, \tau_P\}, P \geq 0$ *de tarefas, define-se* $V_p \subset T \times \Re : \tau \rightarrow v$ *como o conjunto de* **velocidades de processamento** *não-concorrente em* v *das tarefas definidas em* T*.*
- *Dado um conjunto* $U = \{v_0, v_1, \ldots, v_K\}, K \geq 0$ *de unidades de processamento distintas de* v*, define-se* $V_c \subset U \times \Re : v \rightarrow v$ *como o conjunto de* **velocidades médias de comunicação** *de* v *com as unidades de processamento definidas em* U*.*

Definição 7 *Define-se um* **sistema computacional** *como sendo um conjunto* $H = \{v_0, v_1, \ldots, v_P\}, P \geq 0$ *de unidades de processamento.*

Definição 8 *Define-se um* **grupo** G *como sendo um sistema computacional onde existe pelo menos uma unidade de processamento* **coordenadora**, *cuja principal característica é a capacidade de comunicar-se não só com as demais unidades de processamento do próprio sistema como também com unidades de processamento de outros sistemas. As demais unidades de processamento, denominadas de unidades* **trabalhadoras**, *só podem se comunicar com unidades do mesmo sistema heterogêneo.*

Definição 9 *Define-se um* **computador virtual paralelo** *(PVC) como sendo:*

- *Um conjunto* $M = \{G_0, G_1, \ldots, G_Q\}, Q \geq 0$*, de grupos distintos e mutuamente exclusivos, ou seja, que não possuam unidades de processamento em comum.*
- *Um conjunto* $C \subset \aleph^2 : M \times M$*, de conexões não-orientadas entre os grupos de* M*.*

Definição 10 *Define-se um* ***computador virtual maciçamente paralelo*** *(MPVC) como sendo um computador virtual paralelo (PVC) dotado de uma infra-estrutura tal que a adição de uma unidade de processamento qualquer em qualquer um de seus grupos, implique em:*

- *um aumento na performance global do MPVC e/ou*
- *um aumento na robustez do grupo em que esta unidade de processamento foi inserida.*

Por razões de brevidade, neste trabalho será abordada apenas a questão de balanceamento de carga computacional dentro de um único grupo. A apresentação de um esquema para balanceamento de carga entre grupos é reservada para um artigo futuro.

3 A plataforma JOIN

Nascida como fruto de pesquisas iniciadas em 1996 [11] [4] e que, desde então, vem sendo constantemente desenvolvida e aperfeiçoada, a plataforma de software JOIN se propõe a ser um computador vitrual maciçamente paralelo baseado na Internet.

Desenvolvida inteiramente em linguagem Java e dotada de uma estrutura baseada em componentes, a plataforma JOIN ou simplesmente JOIN tem como características sua grande portabilidade, já que pode ser executado em qualquer computador que dispuser de uma implementação da Maquina Virtual Java [7], e sua grande flexibilidade de operação e configuração.

JOIN é um sistema composto por quatro tipos distintos de **componentes**: *servidor, coordenador, trabalhador e jack*. Cada computador participante na plataforma JOIN executa um destes componentes que, dependendo de sua função, poderá estar presente em um ou muitos computadores simultaneamente. Além disso, trabalhadores e coordenadores formam **grupos** de trabalho enquanto que servidores e jacks formam **grupos** especiais.

Como pode ser facilmente deduzido, o conceito de **grupos** em JOIN é facilmente mapeado ao conceito de grupo definido na seção anterior: o computador coordenador e os trabalhadores desempenham exatamente os papéis do coordenador e dos trabalhores introduzidos na definição 8. No entanto, cabe aqui ressaltar que, dentro de um grupo JOIN não é permitida a comunicação direta entre componentes trabalhadores, restrição esta que não é imposta pelo modelo de grupos. A opção por uma topologia $1 : N$ dentro dos grupos JOIN se justifica por questões de escalabilidade.

Os computadores servidor e *jack* podem ser vistos como pertencentes a grupos especiais onde não há trabalhadores, o que garante total compatibilidade da estrutura desta plataforma com aquela exigida pelo modelo de MPVC proposto. Cabe aqui ainda destacar que a maneira como os grupos se interconectam pode ser escolhida arbitrariamente, de acordo com os requisitos do ambiente em que a plataforma será executada.

4 Balanceamento de carga intra-grupo

Em [3] é apresentado um algoritmo cíclico de escalonamento de tarefas em sistemas distribuídos heterogêneos denominado Escalonamento Geracional (*Generational Scheduling* ou GS). Este algoritmo dinâmico utiliza uma heurística bastante simples na fase estática do escalonamento, o que lhe confere baixa complexidade computacional.

Na seção a seguir será apresentado o algoritmo de Escalonamento Geracional com Replicação de Tarefas, que pode ser visto como uma extensão do escalonamento geracional mais adequada ao domínio de interesse deste trabalho, que é o dos MPVCs.

4.1 Escalonador Geracional com Replicação de Tarefas - GSTR

Ambientes virtuais de computação maciçamente paralela baseados na Internet têm como principais características o fraco acoplamento entre as unidades de processamento e seus respectivos coordenadores, e a ampla disponibilidade de unidades de processamento. Uma das implicações dessas duas características é o fato de o sistema estar francamente exposto à ocorrência de falhas em suas UPs.

A ocorrência de falhas, transitórias ou permanentes, nas UPs de um MPVC não deve comprometer o funcionamento da plataforma como um todo. Mais especificamente, a falha em uma UP não deve tornar o estado do sistema inconsistente e não deve impedir a conclusão das aplicações que possuam tarefas em execução nesta UP. Desta forma, um mecanismo de tolerância a falhas deve ser agregado ao sistema de modo a torná-lo robusto à ocorrência de falhas deste tipo.

Há diversas maneiras de se conseguir tolerância a falhas em um sistema distribuído: replicação ativa de componentes, mecanismos de *checkpoint e rollback* e utilização de transações, entre outras [5]. Em um MPVC, o emprego de mecanismos deste tipo, que invariavelmente impõem uma considerável carga sobre o sistema, parece fazer sentido em componentes centralizadores de informações, como os coordenadores e o servidor.

Já em componentes trabalhadores, a utilização de sofisticados mecanismos de recuperação de falhas está vinculada a uma alta taxa *custo / benefício* quando se considera a ampla disponibilidade de componentes deste tipo; a pergunta que se faz é: até que ponto é válida a utilização de mecanismos recuperadores de falhas nos componentes trabalhadores se estes estão disponíveis em grande número na plataforma ?

Diante destes argumentos, parece razoável se esperar que o escalonador de tarefas, que é o módulo responsável pelo gerenciamento das execuções das tarefas, seja capaz de contornar eventuais falhas em UPs às quais tenham sido enviadas tarefas de aplicações, ou seja:

Propriedade 1 *O escalonamento deve ser tolerante a falhas nas unidades de processamento.*

Os escalonadores estático e geracional não prevêem nenhum mecanismo para contornar a ocorrência de falhas nas unidades de processamento e, por isso, não possuem essa propriedade.

Uma segunda implicação do fraco acoplamento das UPs ao MPVC, que se traduz em uma propriedade do escalonador, é:

Propriedade 2 *O escalonamento deve ser capaz de se adaptar a mudanças imprevistas nas cargas das unidades de processamento.*

De fato, em um ambiente como a Internet, na qual as unidades de processamento são computadores voluntariamente cedidos à plataforma e, com isso, não-dedicados ao processamento por ela realizado, mudanças imprevisíveis e significativas em suas cargas são não só possíveis como prováveis. Neste caso, a execução periódica de *benchmarks* para se avaliar em tempo real a carga dessas máquinas é quase sempre impraticável dadas as restrições de comunicação, de escalabilidade e de disponibilidade de poder computacional.

O que se propõe é introduzir um mecanismo de replicação de tarefas ao algoritmo de escalonamento geracional para se introduzir estas duas propriedades. A utilização de replicação de tarefas tem sido largamente explorada pela comunidade científica, porém, raras são as publicações em que este tipo de mecanismo é associado a escalonadores de sistemas paralelos heterogêneos [1] [9]. O algoritmo proposto pode ser descrito da seguinte forma.

1. Na primeira etapa, o problema de escalonamento é formalizado utilizando-se o conjunto de tarefas a serem executadas e as relações de precedência entre elas.

2. Na segunda etapa, todas as tarefas que dependam da execução de outras tarefas são temporariamente ignoradas, reduzindo assim o problema de escalonamento em um problema menor onde não há restrições de precedência. **Tarefas já completadas são também desconsideradas.**

3. Na terceira etapa, um algoritmo de escalonamento (trivial, *best-fit* ou ótimo) é utilizado para a resolução estática do subproblema formalizado na etapa anterior. As tarefas presentes nesta etapa serão aquelas que ainda não foram executadas mas que já se encontram aptas para tal, tarefas previamente escalonadas mas cuja execução ainda não foi iniciada **e tarefas correntemente em execução.**

4. Finalmente, na quarta etapa são detectados eventos de re-escalonamento, que dispararão novamente o algoritmo de maneira a atualizar o estado do problema de escalonamento completo, ou seja, o que considera as relações de precedência entre tarefas.

Como pode ser observado pela descrição apresentada acima, a diferença fundamental entre os algoritmos GS e GSTR reside no fato de neste último a etapa de resolução do subproblema estático considerar também as tarefas correntemente em execução.

Esta diferença garante que a falha em uma unidade de processamento não impedirá que a(s) tarefa(s) que estava(m) em execução nesta unidade seja(m) completada(s) uma vez que, mais cedo ou mais tarde, ela(s) deverá(ão) ser re-escalonada(s) para outra unidade. Também garante que aumentos bruscos nas cargas de unidades de processamento participantes, que afetem significativamente a previsão dos tempos de execução feita pelo escalonador estático, tenham seu impacto atenuado sobre a performance global da execução da aplicação, já que procurará distribuir as tarefas afetadas para outra(s) unidade(s).

5 Testes

Nesta seção serão apresentados os resultados de uma bateria de testes realizados com o objetivo de se verificar as vantagens e desvantagens da proposta de escalonamento intra-grupo introduzida na seção 4.1 frente aos mecanismos estáticos e dinâmicos comumente empregados em sistemas distribuídos.

Para a realização destes testes com a plataforma JOIN, optou-se pela utilização de uma aplicação de busca de números primos segundo o algoritmo da **Peneira de Eratosthenes**. Para resolver este problema, a aplicação determina e particiona o espaço de busca de interesse e promove a busca distribuída de números primos dentro destes intervalos. O emprego deste tipo de processamento se justifica pela grande facilidade com que pode ser configurado e 'por encontrar aplicações em esquemas de teste da robustez de sistemas de criptografia de chave pública.

Esta aplicação é definida por três lotes de tarefas: o primeiro lote, que terá sempre uma única tarefa, é o responsável pelo particionamento do espaço de busca de números primos; o segundo lote, que pode ter multiplicidade variável, é o lote das tarefas que efetuam a busca por números primos dentro de cada um dos intervalos; o terceiro lote, também de multiplicidade 1, é o responsável pela coleta dos resultados gerados pelas tarefas do lote 2.

Foram efetuados testes primeiramente sobre um grupo homogêneo, ou seja, um grupo em que todas as UPs possuem as mesmas características, e em seguida em um grupo heterogêneo, em que as UPs possuem características distintas. Para estes testes, foi utilizado um único grupo de computadores da plataforma.

5.1 Testes sobre um grupo homogêneo

O grupo homogêneo foi formado por 14 computadores do Laboratório de Computação Gráfica da FEEC/Unicamp. Essas máquinas dispõem de processador Pentium IV 1.5 GHz, 512 Mb de memória RAM, sistema operacional Windows 2000 e encontram-se conectadas por uma rede local

Instância	Espaço de Busca ($\times 10^9$)	Tarefas
1	[0..1]	50
2	[0..3]	
3	[0..5]	
4	[0..7]	
5	[0..9]	

Tabela 1. Características das aplicações de teste

Ethernet 10 Mbits/s. Durante a realização destes testes nenhum outro processo de usuário, além dos processos da própria plataforma JoiN, foi executado.

Para estes testes foram geradas 5 instâncias da aplicação de busca de números primos, cada qual parametrizada segundo os valores expressos na Tabela 1.

Como pode ser observado, a diferença entre as instâncias reside unicamente na faixa de valores na qual a busca será efetuada; todas as versões executadas possuiram 50 tarefas, que receberam, cada uma delas, $1/50$ do espaço total de busca.

Para que possa ser possível a avaliação do ganho de performance decorrente da adição de UPs ao sistema, o conjunto formado pelas 5 instâncias da aplicação foi executado sob diferentes cenários: a primeira bateria de execuções foi em um grupo com apenas duas UPs; deste ponto em diante serão sucessivamente adicionadas duas UPs ao sistema e então foi repetida a execução do conjunto de 5 instâncias até chegar ao número máximo de 14 UPs.

Os tempos de execução obtidos podem ser observados na Figura 1.

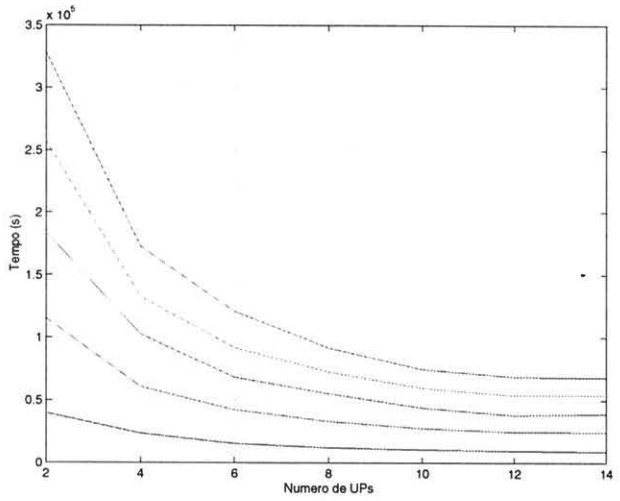

Figura 1. Tempos de execução das 5 instâncias da aplicação de busca de números primos sob diferentes tamanhos do grupo homogêneo.

Neste gráfico pode-se observar que a evolução dos tempos de execução obtidos é bastante regular e previsível. Para que se possa fazer uma avaliação mais completa destes resultados, convém calcular as taxas de *speed-up* e eficiência obtidas.

Para o cálculo de *speed-up* foi desenvolvida uma versão seqüencial da aplicação de busca de números primos. Essa versão seqüencial foi executada em 5 instâncias, parametrizadas identicamente às instâncias geradas para o caso paralelo, em uma das UPs componentes do sistema homogêneo.

Com os tempos das execuções seqüenciais e paralelas obtém-se as curvas de *speed-up* para cada configuração do grupo homogêneo. Essas curvas são mostradas na Figura 2.

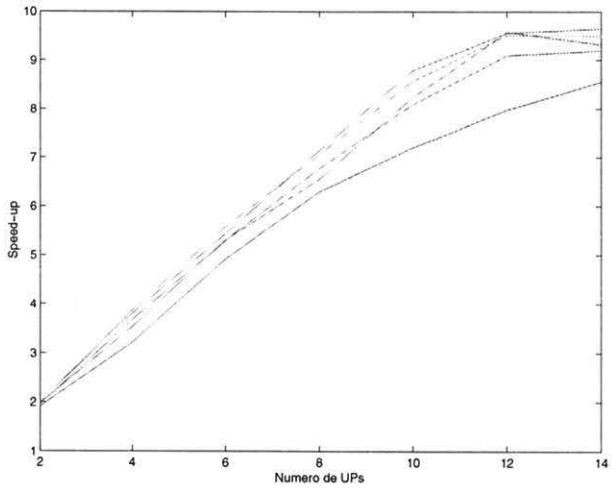

Figura 2. Curvas de *speed-up* para os diferentes tamanhos do grupo homogêneo.

Pode-se observar que a evolução das curvas de *speed-up* inicia-se de forma linear mas satura-se na parte final do gráfico. Essa saturação deve-se tanto às características da plataforma JOIN, que promove a paralelização do processamento, quanto ao suporte físico no qual esta plataforma é executada. Um aspecto importante que pode ser observado é a regularidade na forma das cinco curvas e, como seria de se esperar, o fato de as aplicações com menor relação *computação / comunicação* serem aquelas resultam em menor *speed-up*.

Finalmente, é possível traçar as curvas de eficiência, definida pelo quociente *speed-up / número de UPs*, relativas às execuções paralelas. Essas curvas são apresentadas na Figura 3. Como esperado, as curvas indicam uma saturação no desempenho das execuções, que se acentua na parte final do gráfico. A eficiência média de todas as execuções realizadas neste grupo homogêneo é igual a 84 %.

A seguir serão apresentados os testes realizados sobre grupos heterogêneos.

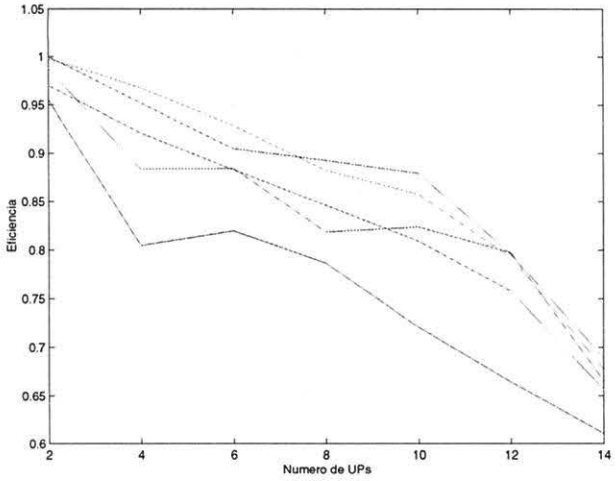

Figura 3. Curvas de eficiêcia para as execuções realizadas sobre o grupo homogêneo.

5.2 Testes sobre grupos heterogêneos

Para os testes sobre grupos heterogêneos, foram utilizados 14 computadores do Laboratório de Computação e Automação da FEEC/Unicamp. O desempenho destes computadores foi avaliado por um conjunto de 30 *benchmarks*, que indicou uma diferença média de performance da ordem de 115 % entre o melhor e o pior computador, evidenciando desta forma a heterogeneidade do sistema. Estes *benchmarks* foram também utilizados para o cálculo de um fator de desempenho médio das UPs participantes que, como será visto mais a frente, é utilizado no cálculo da eficiência.

Nestas testes foram utilizadas apenas 3 das 5 instâncias de aplicações utilizadas no caso homogêneo (as de número 1, 3 e 5). Essa simplificação, necessária por razões práticas, não diminuiu a qualidade dos dados gerados dada a linearidade da complexidade desta aplicação dentro da faixa de valores considerados. Analogamente ao realizado sobre o grupo homogêneo, as UPs utilizadas foram incluídas gradativamente no sistema de modo que o impacto da entrada de novas UPs pudesse ser avaliado. As UPs serão incluídas duas a duas, iniciando-se pelas de melhor desempenho.

Diferentemente do realizado na seção anterior, nesta etapa foram utilizados vários esquemas de escalonamento: assim, foram aplicados os escalonadores estático, GS e GSTR, cada qual com o algoritmo de escalonamento trivial e *best-fit*, perfazendo um total de 6 esquemas de escalonamento.

Em todos os seis casos testados, o desempenho do escalonador variou de maneira uniforme com a mudança da instância da aplicação. Por essa razão e por motivos de brevidade, serão apresentados e comentados somente os tempos de execução obtidos para a última instância (quinta).

A Figura 4 apresenta os tempos de execução obtidos para as escalonamentos estatico, GS e GSTR. A metade da esquerda de cada figura apresenta as curvas obtidas com a utilização do algoritmo de escalonamento trivial, ao passo que a metade da direita ilustra as curvas obtidas com a utilização do algoritmo *best-fit*.

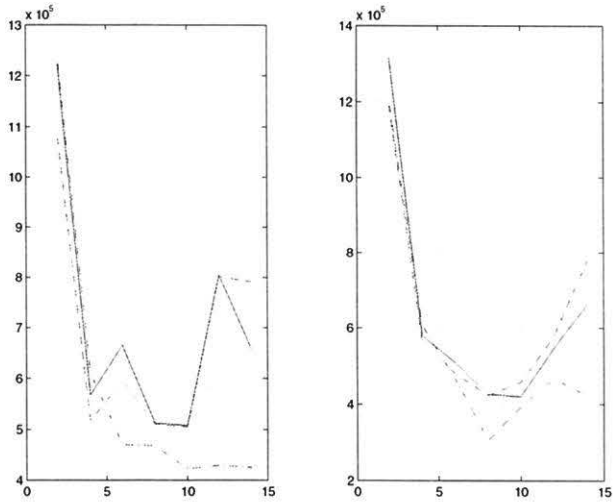

Figura 4. Tempos de execução obtidos com os escalonadores estático (——), GS (- - - -) e GSTR (-·-·-·-) com algoritmos trivial (a esquerda) e *best-fit* (a direita).

Por esta figura pode-se concluir que o comportamento geral dos tempos de execução nos casos considerados é aproximadamente o mesmo. Percebe-se que as curvas tendem a variar de maneira mais suave quando da utilização do algoritmo *best-fit* e que em todos os casos observa-se um grau crescente de saturação no desempenho das execuções à medida que cresce o número de UPs.

Para o cálculo dos *speed-ups* obtidos nos grupos heterogêneos, foi executada na máquina de melhor desempenho do grupo a versão seqüencial da aplicação de números primos parametrizada segundo as três instâncias utilizadas nestes testes. Os gráficos da Figura 5 mostram as taxas de *speed-up* obtidas pela utilização dos escalonadores estático, GS e GSTR com algoritmos trivial (a esquerda) e *best-fit*.

Todos estes gráficos possuem um comportamento bastante similar; neles fica clara a melhora em desempenho apresentada pelo escalonador GSTR à medida que o sistema torna-se cada vez mais heterogêneo.

Uma análise adicional que pode ser feita sobre estes dados é o cálculo da eficiência com que a aplicação de busca de números primos foi paralelizada e executada neste grupo heterogêneo. Para isso, foi utilizada a expressão de eficiência em sistemas heterogêneos extraída de [6], onde

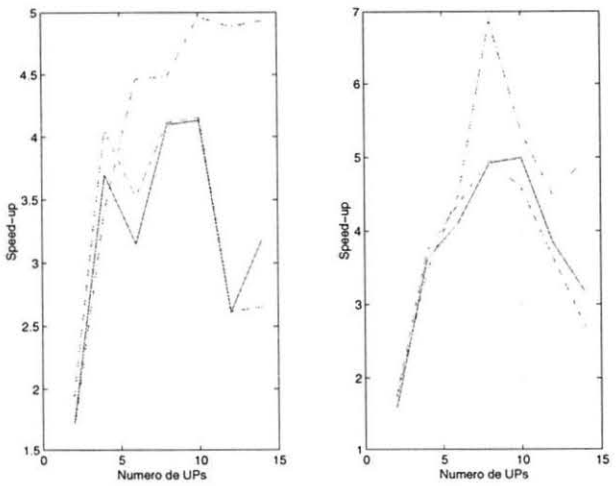

Figura 5. Taxas de *speed-up* obtidas pela utilização dos escalonadores estático (——), GS (- - - -) e GSTR (·-·-·-·) com algoritmos trivial (a esquerda) e *best-fit* (a direita).

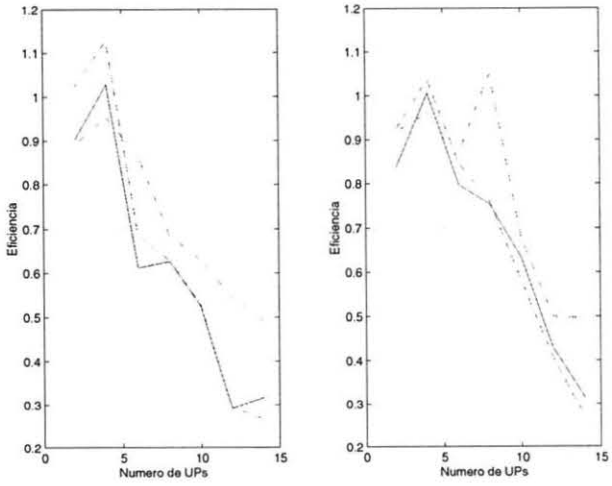

Figura 6. Taxas de eficiência obtidas pela utilização dos escalonadores estático (——), GS (- - - -) e GSTR (·-·-·-·) com algoritmos trivial (a esquerda) e *best-fit* (a direita).

Escalonador	Algoritmo	Eficiência média
Estático	Trivial	60.41 %
Estático	Best-fit	67.10 %
GS	Trivial	63.14 %
GS	Best-fit	67.46 %
GSTR	Trivial	69.20 %
GSTR	Best-fit	71.19 %

Tabela 2. Taxas médias de eficiência obtidas para cada esquema de escalonamento.

o cálculo da eficiência do sistema é ponderado pelo grau de heterogeneidade de suas unidades de processamento. Neste caso, os pesos uilizados foram os fatores de desempenho calculados a partir dos *benchmarks* executados inicialmente.

Os gráficos da Figura 6 mostram as taxas de eficiência obtidas pela utilização dos três escalonadores com algoritmos trivial e *best-fit*.

Analogamente ao ilustrado pelas curvas de *speed-up*, pode-se perceber pelos gráficos de eficiência a maior regularidade na evolução dos tempos obtidos pela utilização do algoritmo *best-fit*, assim como o melhor desempenho global fornecida pelo escalonador GSTR frente aos demais. Pode-se ainda através desses dados calcular a eficiência global de cada esquema de escalonamento para a bateria de teste propostos. As eficiências globais, calculadas como sendo a média aritmética de todos os valores encontrados para um determinador esquema de escalonamento, podem ser vistas na Tabela 2.

5.3 Resultados e Conclusões

Nestes testes, foi possivel verificar que:

1. a utilização do algoritmo de escalonamento *best-fit* proporcionou maior regularidade na evolução dos tempos de execução e melhor performance global;
2. o escalonador GSTR apresentou melhor performance global que os demais, sobretudo à medida que o sistema tornou-se mais heterogêneo;
3. o escalonador GSTR mostrou-se bastante robusto à introdução de heterogeneidade no sistema.

Em linhas gerais, os testes indicaram que o emprego de um escalonador GSTR com escalonamento *best-fit* leva a resultados melhores que o de outros esquemas considerados. Além de trabalhar em menor tempo e de forma mais estável, este escalonador possui a vantagem única de ser tolerante a falhas nas unidades de processamento.

Apesar dos outros esquemas de escalonamento terem apresentado resultados ligeiramente melhores em sistemas pouco heterogêneos, e onde supostamente a tolerância a falhas não era um requisito fundamental, cabe aqui destacar que dentro do domínio de interesse deste trabalho, que é o dos computadores virtuais maciçamente paralelos (MPVCs), o esquema de escalonamento proposto destacou-se claramente frente aos demais.

6 Trabalhos futuros

Dentre os principais pontos a serem desenvolvidos futuramente, destacam-se os seguintes (em ordem decrescente de importância).

1. Implementação e teste de algoritmo (já desenvolvido) para balanceamento de carga entre grupos.
2. Modificação da implementação em JOIN do algoritmo GSTR para que tarefas isoladas que já tenham seus dados de entrada disponíveis sejam iniciadas. Atualmente, as tarefas de um lote são iniciadas somente quando os dados de entrada de todas as tarefas deste lote já se encontram disponíveis.
3. Incluir mecanismos para o salvamento do estado dos componentes servidor e coordenador dos serviços gerenciador de aplicações e escalonador para permitir a recuperação de falhas.
4. Introdução de um mecanismo para a execução de aplicações com diferentes prioridades.

Referências

[1] L. Almand and Y. K. Kwok. A new approach to scheduling parallel programs using task duplication. In *International Conference on Parallel Processing*, volume II, pages 47–51, 1994.

[2] Edward K. Blum, Xin Wang, and Patrick Leung. Architectures and message-passing algorithms for cluster computing: Design and performance. *Parallel Computing*, 26:313–332, 2000.

[3] Brent R. Carter, Daniel W. Watson, Freund Richard F., Keith Elaine, Mirabile Francesca, and Howard Jay Siegel. Generational scheduling for dynamic task management in heterogeneous computing systems. *Journal of Information Sciences*, 106:219–236, 1998.

[4] Marco A. Amaral Henriques. A proposal for a Java based massively parallel processing on the web. In *Proceedings of The First Annual Workshop on Java for High-Performance Computing*, pages 56–66, 1999.

[5] Pankaj Jalote. *Fault Tolerance in Distributed Systems*. Prentice Hall, 1994.

[6] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamim/Cummings, 1994.

[7] Sun Microsystems. The Java Virtual Machine Specification. Documentação on-line: http://java.sun.com/docs/books/vmspec/index.html, 2002.

[8] V. S. Sunderam. Pvm: A framework for parallel distributed computing. *Concurrency, Practice and Experience*, 2(4):315–340, 1990.

[9] Tatsuhiro Tsuchiya, Tetsuya Osada, and Tohru Kikuno. Genetics-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems*, 22:197–207, 1998.

[10] Thomas M. Warschko, Walter F. Tichy, and Christian G. Herter. Efficient parallel computing on workstation clusters. *PARS (GI) Mitteilungen*, 14:49–57, 1995.

[11] Eduardo Javier Huerta Yero. Um sistema para o processamento massivamente paralelo na world wide web. Master's thesis, Faculdade de Engenharia Elétrica e de Computação - Unicamp, 1998.