# I Escola Regional de Alto Desempenho de SP
# Fórum de Pós-Graduação

ERAD-SP 2010

30 e 31 de Julho de 2010

# Mensagem dos Coordenadores do Fórum de Pós-Graduação

O Fórum de Pós-Graduação é parte da Escola Regional de Alto Desempenho de São Paulo (ERAD-SP). Seu objetivo é criar um espaço para a divulgação de trabalhos desenvolvidos em nível de pós-graduação na área de Processamento de Alto Desempenho, Arquitetura de Computadores e Sistemas Distribuídos no Estado de São Paulo. Esperamos que essa iniciativa aproxime os pesquisadores, atraia o interesse de novos alunos, suscite novas colaborações e que fortaleça a pesquisa nesses temas no Estado. Esperamos criar um espaço aberto para apresentação e discussão de idéias e trabalhos no seus diversos estágios de desenvolvimento, que estimule a produção científica do Estado de São Paulo em outros fóruns nacionais e internacionais.

Hermes Senger (UFSCAR)
Rodolfo Jardim de Azevedo (UNICAMP)

## Coordenação Geral

Denise Stringhini (Mackenzie)

## Coordenação de Programa

Alfredo Goldman (USP)

## Coordenação do Fórum de Pós-Graduação

Hermes Senger (UFSCAR)
Rodolfo Jardim de Azevedo (UNICAMP)

## Coordenação da Sessão de Iniciação Científica

Raphael Y. de Camargo (UFABC)

## Coordenação do Open Contest

Augusto Gomes Jr. (USP)
Calebe de Paula Bianchini (Mackenzie)

## Coordenação Financeira e de Patrocínio

Francisco Masseto (UFABC)

## Comitê Organizador

Alfredo Goldman (USP)
Augusto Gomes Jr. (USP)
Calebe de Paula Bianchini (Mackenzie)
Denise Stringhini (Mackenzie)
Edson Midorikawa (USP)
Francisco Masseto (UFABC)
Hermes Senger (UFSCAR)
José Márcio Machado (UNESP)
Liria Matsumoto Sato (USP)
Luciano Silva (Mackenzie)
Nicolas Kassalyas (São Judas)
Raphael Y. de Camargo (UFABC)
Roberto Kenji (USP)
Rodolfo Jardim de Azevedo (UNICAMP)
Sérgio Takeo Kofuji (USP)

## Comitê de Programa

Aleardo Manacero Jr. (IBILCE - UNESP)
Alfredo Goldman (IME - USP)
Augusto Gomes Jr (Anhembi Morumbi)
Calebe de Paula Bianchini (Mackenzie)
Celso Hirata (ITA)
Denise Stringhini (Mackenzie)
Edson Midorikawa (Poli - USP)
Francisco Massetto (UFABC)
Hélio Guardia (UFSCar)
Hermes Senger (UFSCar)
José Machado (UNESP)
Liria Sato (Poli - USP)
Marcos Cavenaghi (UNESP)
Marcos Santana (ICMC - USP)
Raphael Camargo (UFABC)
Regina Santana (ICMC - USP)
Renata Spolon Lobato (UNESP)
Roberta Spolon (UNESP)
Rodolfo Azevedo (UNICAMP)
Rodrigo Mello (ICMC - USP)
Sergio Kofuji (POLI - USP)
Siang Song (IME - USP)

# Índice


- Tolerância a Falhas
  - Caracterização de Comportamento de Sistemas por meio de Agrupamento de Dados e Detecção de Novidades 1
    Eduardo Alves Ferreira, Rodrigo Mello
  - Um Detector de Faltas Baseado no Comportamento de Processos 5
    Cássio Pereira, Rodrigo Mello
  - Fault Tolerance using Redundant Gateway Protocols 9
    Alessandro Kraemer, Kaio Vilar, Alfredo Goldman
  - Tolerância a falhas no armazenamento distribuído de dados em grades oportunista 13
    Pablo Laura-Huaman, Rodrigo Mello
- Computação em Nuvem
  - Memory Dispatcher: Gerência de Recursos em Ambientes Virtualizados 18
    Artur Baruchi, Edson Midorikawa
  - Arquitetura de Plataforma para Computação em Nuvem visando a Interoperabilidade 22
    Charles Rodamilans, Edson Midorikawa
  - Caracterização de aplicações científicas e transacionais em ambientes Cloud Computings 26
    Denis Ogura, Edson Midorikawa
- Alto Desempenho
  - Análise de Escalabilidade e Determinação de Número Adequado de Processos para Aplicações Paralelas em Grids 32
    Samuel Silva, Hélio Guardia
  - Atenuação do Gargalo no Acesso a Disco em Sistemas Multiprocessadores 36
    Darlon Vasata, Liria Sato
  - Finding Fractional Identities in Multiple Sequences Using a Fast Parallel Algorithm 40
    Evandro Marucci, Geraldo Zafalon, Aleardo Manacero Jr., Liria Sato, José Machado
  - Alocação de recursos distribuídos em Grade de computadores 44
    Francisco Ribacionka, Liria Sato, Luciana Arantes
- Computação em Grade
  - Acesso e Compartilhamento de Bancos de Dados Heterogêneos Integrados Através de Grade Computacional 50
    Fernando Kakugawa, Liria Sato, Mathias Brito

– Implementação da Interface MPI e de sua Infra-estrutura para Grades Computacionais 54
Augusto Gomes Jr, Liria Sato, Calebe Bianchini, Francisco Massetto, Nilton Paula
– Uma Abordagem Orientada a Sistemas para Otimização de Escalonamento de Processos em Grades Compucionais 58
Paulo Gabriel
– Escalonamento de Workflows com Tarefas Paralelas e Seqüenciais em Grades Computacionais 62
Silvio Stanzani, Liria Sato, Nilton Paula

# Sessão 1

# Tolerância a Falhas

# Caracterização de Comportamento de Sistemas por meio de Agrupamento de Dados e Detecção de Novidades

**Eduardo Alves Ferreira**[1], **Rodrigo Fernandes de Mello**[1]

[1] Instituto de Ciências Matemáticas e de Computação – USP
Caixa Postal: 668 – 13.560-970 – São Carlos – SP – Brasil

{eaf,mello}@icmc.usp.br

***Abstract.*** *Systems behavior characterization is a commonly used technique to perform process scheduling, fault detection and intrusion prevention. This work describes an approach to system behavior characterization based on clustering and novelty detection, which allows the representation of unstructured contexts. This approach is evaluated using a system characterization dataset, in which we distinguish normal from anomalous scenarios and observed that the precision of detection is directly proportional to the quality of the clustering stage.*

***Resumo.*** *A caracterização de comportamento de sistemas é uma técnica frequentemente utilizada para escalonamento de recursos, detecção de faltas e prevenção de intrusões. Este trabalho descreve uma proposta para a caracterização do comportamento de sistemas baseada em agrupamento de dados e detecção de novidades, que permite a caracterização de contextos pouco estruturados de aplicações. A proposta é avaliada com o uso de um conjunto de dados de comportamento de processos, onde se observa que situações anômalas são diferenciáveis de situações normais e que a qualidade da detecção é diretamente proporcional à qualidade de agrupamento.*

## 1. Introdução

A caracterização de comportamento de processos ou fluxos de dados em uma aplicação é uma técnica frequentemente utilizada para diversas tarefas, como otimização de escalonamento de processos, otimização de acesso a dados [Dodonov e de Mello 2010, Ishii e de Mello 2009], ou detecção de faltas e intrusões [Pereira e de Mello 2009, Twycross e Aickelin 2010, Forrest et al. 1996]. Diversos trabalhos utilizam uma abordagem supervisionada para detecção de falhas ou intrusões, na qual o comportamento normal de uma aplicação (i.e. o comportamento durante uso esperado e não malicioso da aplicação) é extraído durante o funcionamento em ambiente isolado ou simulado. Em seguida, monitora-se o comportamento da aplicação em ambiente de produção, buscando por novidades que são associadas a faltas ou intrusões.

Alguns trabalhos monitoram aspectos unidimensionais de uma aplicação, gerando séries temporais nas quais técnicas de detecção de novidades são aplicadas a fim de caracterizar o comportamento do sistema. Esses trabalhos se mostraram válidos na caracterização do comportamento de aplicações UNIX visando detecção de faltas e intrusões, apresentando precisão suficiente para a distinção de situações normais e anômalas. Contudo, essas técnicas não são aplicáveis em contextos pouco estruturados, que se apresentam, por exemplo, no monitoramento de diversas variáveis de uma

aplicação, de parâmetros de requisições *Web*, em comandos executados em aplicações de infraestrutura (como SGBD's ou *Shells*) ou linhas de *log*. O objetivo deste trabalho é a detecção de injeções de código em aplicações *Web*, e para isso propõe-se a aplicação de uma etapa de quantização sobre dados pouco estruturados, a fim de gerar uma série temporal que represente o comportamento da aplicação. Essa quantização é feita por meio de técnicas de agrupamento de dados, o que permite a definição de uma série temporal a partir de um contexto pouco estruturado, demandando apenas a definição de uma função de distância entre objetos.

O uso dessa técnica é validado por meio de um conjunto de dados de caracterização de comportamento de aplicações [Twycross e Aickelin 2010], que representa o comportamento de aplicações UNIX em situações normais e de ataque. Verificou-se que diversos algoritmos *on-line*[1] de agrupamento de dados e detecção de novidades apresentam precisão compatível com algoritmos *batch* para esse conjunto de dados. Por fim, é analisada a correlação entre os índices de validação de agrupamento e a separação entre situações normais e de ataque, onde se observa que agrupamentos com maiores índices de qualidade geram séries contendo maior quantidade de informação, o que permite uma caracterização mais precisa do comportamento do sistema.

## 2. Arquitetura proposta

A arquitetura proposta é apresentada na Figura 1. Nessa arquitetura, monitoram-se os aspectos da aplicação que sejam relevantes ao problema em questão (e.g. chamadas de sistema, uso de CPU ou memória, mensagens de rede ou linhas de *log*). Deve-se então definir uma função de distância entre observações, e escolher um algoritmo de agrupamento e de detecção de novidades.

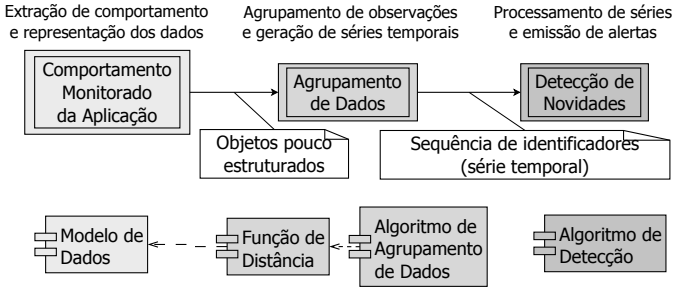



**Figura 1. Arquitetura proposta**

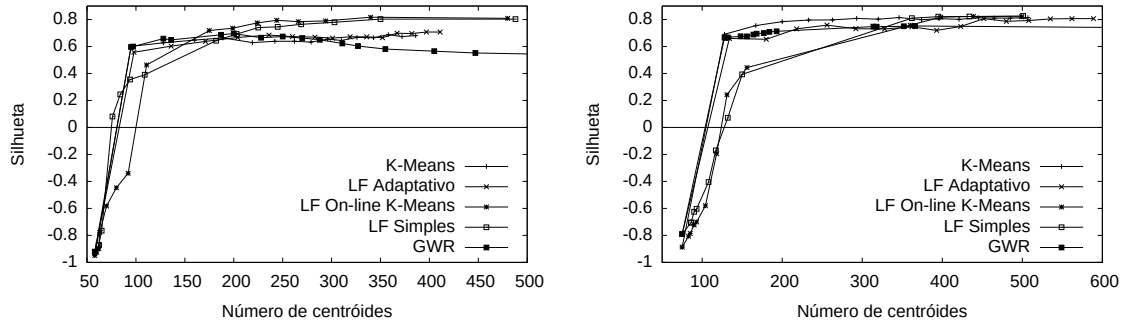
Diversos algoritmos de agrupamento e detecção de novidades foram aplicados ao conjunto de dados de chamadas de sistema[2] fornecido por Twycross e Aickelin [2010]. Esse conjunto de dados contém as chamadas de sistemas de duas aplicações UNIX (`rpc.statd`, monitor do protocolo RPC e `WU-FTPD`, servidor FTP) em sessões de uso normal e em situações de ataque. Foi definida uma função de distância entre chamadas de sistema que considera os valores dos parâmetros das chamadas, e o algoritmo de agrupamento une chamadas de mesmo tipo considerando essa distância. Essa abordagem garante que, quando se obtém menor qualidade de agrupamento (i.e. quando

[1]É necessário que se utilizem algoritmos *on-line* uma vez que não é viável armazenar informações sobre todo o funcionamento de aplicações em nenhum tipo de memória.

[2]Disponível em `http://www.cs.nott.ac.uk/~jpt/datasets.html`

todos os objetos são agrupados em um único conjunto), a série gerada contém apenas informações sobre o nome da chamada, de forma idêntica aos trabalhos anteriores [Twycross e Aickelin 2010, Forrest et al. 1996]. Conforme a qualidade de agrupamento aumenta, uma maior quantidade de informação é agregada à série, de forma que seus pontos passam a representar não apenas o tipo das chamadas de sistema mas também os valores de seus parâmetros.

Os algoritmos *on-line* de agrupamento de dados avaliados foram o *Leader-Follower* Simples, *Leader-Follower* com *On-line K-Means*, *Leader-Follower* Adaptativo e uma adaptação *on-line* da rede neural GWR [Xu e Wunsch 2008, Charikar et al. 1997, Marsland et al. 2002]. Além disso, o algoritmo *K-Means* foi aplicado para comparação da precisão com algoritmos *on-line*. A Figura 2 apresenta o valor da silhueta [Rousseeuw 1987] para ambos os conjuntos de dados. Observa-se pouca variação na qualidade de agrupamento entre as abordagens *On-line*, além de precisão compatíveis com a do algoritmo *K-Means*, principalmente quando o número de centróides é elevado.




**(a)** Conjunto `rpc.statd` **(b)** Conjunto `WU-FTPD`

**Figura 2. Qualidade de agrupamento**

Os algoritmos de detecção de novidades avaliados foram o de Janela Deslizante, Entropia da Cadeia de Markov e, para fins de comparação, a distância DTW [Forrest et al. 1996, Pereira e de Mello 2009]. A Tabela 1 apresenta a quantidade de novidade observada em situações normais e de ataque, em situações de qualidade mínima ($k = 1$) e máxima ($k = 16$) de índices de agrupamento. Observa-se um aumento significativo na diferença das médias de situações normais e de ataque com o aumento da qualidade de agrupamento. A diferença entre essas médias é utilizada como índice de qualidade de detecção de anomalias.

**Tabela 1. rpc.statd: DTW / K-Means.**

| | $k = 1$ | | | | $k = 16$ | | | |
|---|---|---|---|---|---|---|---|---|
| | normal 1 | normal 2 | ataque 1 | ataque 2 | normal 1 | normal 2 | ataque 1 | ataque 2 |
| normal 1 | 0.00% | 1.52% | 36.14% | 36.17% | 0.00% | 1.71% | 53.06% | 53.14% |
| normal 2 | 1.52% | 0.00% | 36.08% | 36.14% | 1.71% | 0.00% | 52.67% | 52.72% |

A Tabela 2 apresenta o incremento nos índices de anomalia quando ocorre o aumento na qualidade de agrupamento (i.e. comparam-se as situações onde a silhueta é mínima e máxima). Observa-se que o aumento na qualidade de diferenciação é significativo (com intervalo de confiança de 95%) para todos os algoritmos de agrupamento e

detecção. Observa-se, também, alta correlação (com média na ordem de $0,80$) entre os índices de qualidade de agrupamento de dados e de detecção de anomalias.

**Tabela 2. rpc.statd:Aumento na separação.**

| | LF Adaptativo | GWR | LF Simples | K-Means | LF On-line K-Means |
|---|---|---|---|---|---|
| Entropia | 18.44 | 19.67 | 31.53 | 18.18 | 31.51 |
| Janela Deslizante | 15.53 | 20.73 | 16.69 | 19.83 | 15.75 |
| DTW | 16.95 | 16.69 | 16.94 | 16.67 | 16.92 |

## 3. Conclusões e trabalhos futuros

Este trabalho apresentou uma abordagem para a caracterização de comportamento de sistemas que demanda pouco conhecimento sobre sua estrutura e permite uso em contextos pouco estruturados. Essa abordagem pode ser utilizada para a detecção de faltas ou intrusões em sistemas computacionais, tendo sido validada em um conjunto de dados de comportamento de processos no sistema UNIX. Como próximos passos pretende-se avaliar a aplicabilidade dessa técnica em contexto ainda menos estruturados (como na observação de valores de variáveis de aplicações ou parâmetros de requisições *Web*), além de avaliar técnicas bioinspiradas (como, por exemplo, sistemas imunológicos artificiais) para a etapa de agrupamento de dados.

## Referências


Charikar, M., Chekuri, C., Feder, T., e Motwani, R. (1997). Incremental clustering and dynamic information retrieval. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, páginas 626–635.

Dodonov, E. e de Mello, R. F. (2010). A novel approach for distributed application scheduling based on prediction of communication events. *Future Gener. Comput. Syst.*, 26(5):740–752.

Forrest, S., Hofmeyr, S. A., Somayaji, A., e Longstaff, T. A. (1996). A sense of self for unix processes. *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, páginas 120–128.

Ishii, R. P. e de Mello, R. F. (2009). A history-based heuristic to optimize data access in distributed environments. *The 21st IASTED International Conference on Parallel and Distributed Computing and Systems*, 1:1–8.

Marsland, S., Shapiro, J., e Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, páginas 1041–1058.

Pereira, C. M. M. e de Mello, R. F. (2009). Behavioral study of unix commands in a faulty environment. *Proceedings of the 8th International Conference on Dependable, Autonomic and Secure Computing*, páginas 1–6.

Rousseeuw, P. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, páginas 53–65.

Twycross, J. e Aickelin, U. (2010). Information fusion in the immune system. *Information Fusion, Volume 11, Issue 1*, páginas 35–44.

Xu, R. e Wunsch, D. (2008). *Clustering (IEEE Press Series on Computational Intelligence)*. Wiley-IEEE Press, 1 edição.

# Um Detector de Faltas Baseado no Comportamento de Processos

**Cássio M. M. Pereira, Rodrigo F. de Mello**

[1]Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo (USP)
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brasil

{cpereira,mello}@icmc.usp.br

***Abstract.*** *Recently, there has been an increased interest in self-healing systems. Those types of systems use fault detection techniques to prevent failures. The techniques proposed in the literature have been based on three mainstream approaches: process heartbeats, statistical analysis and machine learning. However, these approaches present several limitations, such as not dealing directly with faults and not modeling non-linear, causally dependent behavior. The proposed approach overcomes these limitations by detecting faults and considering the non-linear, causally related behavior of processes through Markov chains and Radial Basis Functions. Experiments confirm the advantages of the proposed approach.*

***Resumo.*** *Recentemente, houve um aumento no interesse por sistemas autocuráveis. Tais sistemas utilizam técnicas de detecção de faltas para prevenir falhas. As técnicas propostas na literatura têm sido baseadas em três áreas: monitoramento de heartbeats, estatística e aprendizado de máquina. Entretanto, essas abordagens têm limitações, como não tratar faltas diretamente, não modelar comportamento não linear e causalmente dependente. A abordagem proposta supera essas limitações detectando faltas e considerando comportamento não linear e dependente de processos, por meio de cadeias de Markov e funções de base radial. Experimentos confirmam as vantagens da abordagem proposta.*

## 1. Introdução

Recentemente, o conceito de computação nas nuvens (Armbrust et al. 2009) renovou o interesse pela interconexão de computadores, os quais compõem aglomerados ou grades. Aglomerados são, geralmente, caracterizados por computadores de arquitetura homogênea, interconectados em uma rede local de alta velocidade (Kon e Goldman 2008). Grades são compostas por recursos de *hardware* e *software* heterogêneos e geograficamente distribuídos, interconectados por redes de diferentes características (Tanenbaum e Steen 2007). Essas tecnologias são, inerentemente, voltadas para aplicações específicas, de grande porte e que buscam, essencialmente, por alto desempenho e alta disponibilidade. No entanto, para atender ambos requisitos, faz-se necessário prover consistência, segurança, escalonamento eficiente, tolerância a faltas, etc.

Entre esses aspectos, tolerância a faltas tem sido o foco de diversas pesquisas (Xue et al. 2007; Filho et al. 2008). Para compreender as pesquisas existentes na área é

importante primeiramente definir seus principais termos. Neste trabalho adotou-se a taxonomia proposta por (Gärtner 1999), que define os seguintes termos: **falta** é uma transição inesperada entre estados do sistema; **erro** é um estado fora da especificação do sistema; e **falha** é um erro em efeito, no qual a saída do sistema diverge de sua especificação. Nesse contexto, diz-se que uma falta causa um erro, o qual pode, por sua vez, ocasionar uma falha no sistema. Diversas abordagens têm buscado detectar faltas. Elas podem ser divididas em três frentes básicas: abordagens baseadas em *heartbeat*, estatística e aprendizado de máquina. Na seção seguinte apresenta-se trabalhos existentes nessas frentes.

## 2. Trabalhos Relacionados

As abordagens baseadas em *heartbeats* (van Renesse et al. 1998; Filho et al. 2008) foram concebidas para utilização em sistemas distribuídos. Elas funcionam por meio de um monitor que recebe, periodicamente, informações sobre o estado dos processos monitorados. Caso um intervalo de tempo $\delta t$ transcorra sem o recebimento de um "*heartbeat*", o processo monitorado passa a ser considerado falho. Uma desvantagem das técnicas puramente baseadas em *heartbeats* é que elas detectam somente falhas, ou seja, situações em que os processos monitorados já falharam e pararam de responder. Uma análise mais detalhada dos processos monitorados pode fornecer mais indícios sobre a situação de cada um deles. Nesse contexto, foram propostas abordagens que visam aprender características de um processo durante sua execução. Essa abordagens utilizam conceitos estatísticos e de aprendizado de máquina.

Técnicas baseadas em conceitos estatísticos (Li et al. 2002; Xue et al. 2007) têm empregado modelos auto-regressivos (AR), de médias móveis (MA), auto-regressivos de médias móveis (ARMA) e auto-regressivos integrados de médias móveis (ARIMA), para modelar o perfil de sistemas, a fim de detectar momentos em que o sistema apresenta comportamento fora do esperado, como por exemplo falta de memória causada por alocação incorreta. No entanto, essas técnicas são capazes de modelar somente comportamentos lineares, o que levou à proposta de técnicas baseadas em aprendizado de máquina, as quais podem modelar comportamentos mais complexos.

Abordagens baseadas em aprendizado de máquina (Turnbull e Alldrin 2003; Hoffmann et al. 2007) têm utilizado técnicas como redes neurais RBF (*Radial Basis Function*) e *Support Vector Machines* (SVM), devido à sua capacidade de modelar comportamentos não lineares e seus bons resultados em diversas áreas (Lorena e Carvalho 2007). Entretanto, essas técnicas assumem, geralmente, que os dados analisados são gerados independente e identicamente distribuídos. Essa suposição nem sempre corresponde à realidade, especialmente no cenário de detecção de faltas, já que uma falta provavelmente vai afetar o comportamento subsequente do sistema.

As técnicas existentes para detecção de faltas, nas subáreas de estatística e aprendizado de máquina, também não consideram variações no comportamento dos processos que geraram as séries analisadas. As chamadas de sistema que o processo utiliza, juntamente com atributos e retorno dessas funções não são empregados, o que resulta na perda de informações relevantes para a detecção de faltas. Observando as limitações nas pesquisas relacionadas foi proposta uma nova abordagem que leva em conta o comportamento dos processos e é capaz de modelar sua dependência e não linearidade.

## 3. Abordagem Proposta

A abordagem proposta considera redes neurais com funções de base radial, cadeias de Markov e Entropia, para detectar novidades no comportamento de processos. Nesse contexto, novidades podem corresponder a estados errôneos visitados, os quais são diferentes do comportamento normal esperado.

O primeiro passo da abordagem é computar a distância Euclidiana de cada padrão de entrada aos centróides já criados da rede, os quais representam informações já conhecidas, ou o comportamento normal modelado. A distância é calculada utilizando a função $D(\cdot)$ (Equação 1), que considera o padrão de entrada ($I$) e os centróides ($C$). Na equação, $I_{ik}$ e $C_{jk}$ são atributos do padrão de entrada $i$ e do centróide $j$, respectivamente, e $n$ é o número de atributos.

$$D(I_i, C_j) = \sqrt{\sum_{k=1}^{n}(I_{ik} - C_{jk})^2}, \quad Act(x, \sigma) = e^{-\frac{x^2}{2\sigma^2}}, \quad E_t = -\sum_{j=1}^{c}\sum_{i=1}^{c} p(i,j) \cdot log_2(p(i,j)) \qquad (1)$$

As distâncias dos exemplos de treinamento a cada centróide são utilizadas como entrada para as funções de base radial (RBF), as quais calculam a ativação de cada centróide à entrada. Este trabalho considera a função $Act$ (Equação 1) para computar a ativação, a qual segue uma função Gaussiana. Após computar a ativação, o valor obtido é comparado com um nível de aceitação, denominado *threshold*. Esse parâmetro define o nível mínimo de ativação que um neurônio precisa atingir para aceitar o padrão como pertencente a si. A ativação de todos os neurônios é então calculada. Aquele que fornece a maior ativação é selecionado como o receptor do padrão de entrada. Isso significa que o padrão será agrupado com esse neurônio. Caso nenhum neurônio apresente ativação suficiente, cria-se um novo, tendo como centróide os valores do padrão de entrada.

Cada neurônio da rede RBF é considerado como um estado da cadeia de Markov. As transições entre estados são armazenadas em uma matriz de probabilidades, que é usada para representar a relação entre estados que um processo visita durante sua execução. Por exemplo, considere um conjunto de dados que contenha os seguintes atributos: número de *bytes* lidos/escritos em disco ($x_1$), memória disponível ($x_2$) e espaço em disco disponível ($x_3$). Um estado $s$ desse processo pode ser representado por uma tupla $s = (x_1, x_2, x_3)$.

Após obter a cadeia de Markov para os padrões de entrada apresentados, a Entropia ($E_t$) do sistema no tempo $t$ é calculada de acordo com a Equação 1, a qual considera todas as probabilidades de transição. Nessa equação, $p(i, j)$ representa a probabilidade de transição do estado $s_i$ para $s_j$, sendo $c$ o número de centros. Considera-se $0 \cdot log_2(0) = 0$.

Essa abordagem tem como saída o quadrado da variação da Entropia entre padrões de entrada consecutivos. Quando um novo estado é visitado, significa que um neurônio acabou de ser criado e certo nível de novidade é detectado. Isso causa um aumento na Entropia do sistema, que é refletido na saída da abordagem.

## 4. Experimentos

Foram realizados experimentos para avaliar a eficácia da abordagem na detecção de faltas. Foram injetadas faltas em três utilitários Linux: no programa `cp` para cópia de arquivos,

**Tabela 1. Resultados dos experimentos**

| Medida de Avaliação | Abordagem Proposta | | | SVM | | | ARIMA | | |
|---|---|---|---|---|---|---|---|---|---|
| | tar | cp | rsync | tar | cp | rsync | tar | cp | rsync |
| **Taxa Verdadeiro Positivo** | **0.94** | **0.68** | **1.00** | 0.26 | 0.54 | 0.01 | 0.00 | 0.00 | 0.27 |
| **Taxa Falso Positivo** | 0.74 | **0.00** | 0.40 | 0.21 | 0.36 | **0.15** | **0.02** | 0.01 | 0.39 |
| **F-Measure** | **0.79** | **0.81** | **0.28** | 0.24 | 0.56 | 0.02 | 0.00 | 0.01 | 0.08 |

no programa `tar` para arquivamento e compressão de dados e no programa `rsync`, que permite sincronizar arquivos com uma máquina remota. As faltas foram injetadas por meio da interceptação de chamadas de sistema e modificação de registradores, sendo a *system call* `ptrace` utilizada para esse fim. O tempo médio entre faltas foi simulado seguindo uma distribuição *Weibull*. A Tabela 1 apresenta os resultados obtidos e uma comparação com as técnicas ARIMA e SVM. Medidas mais altas para a Taxa de Verdadeiros Positivos e F-Measure (Witten e Frank 2005) representam maior eficácia, enquanto uma menor taxa de Falsos Positivos é melhor. Os melhores resultados para cada experimento, considerando as três abordagens, estão assinalados em negrito.

### 4.1. Conclusões

Os bons resultados obtidos com a abordagem devem-se, principalmente, a três fatores. O primeiro é utilizar informações sobre o comportamento do processo, tais como suas chamadas de sistema e argumentos passados. O segundo é não assumir que os dados foram gerados de maneira i.i.d., ou seja, de forma idêntica e de amostras independentes. Isso faz com que sejam consideradas as relações causais entre observações temporais dos processos analisados. A cadeia de Markov permite capturar esse relacionamento causal ao longo do tempo. Além disso, o terceiro fator é não assumir modelos lineares, o que é conseguido pelo emprego de funções de base radial, que permitem modelar a distribuição dos dados como uma gaussiana.

## Agradecimentos

Este trabalho foi financiado incialmente pelo CNPq e atualmente é financiado pela FAPESP, processo n° 2009/04645-0.

## Referências

[Armbrust et al. 2009] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*.

[Filho et al. 2008] Filho, F. C., Marques, A., de Camargo, R. Y., e Kon, F. (2008). A group membership service for large-scale grids. In *MGC '08: Proceedings of the 6th international workshop on Middleware for grid computing*, pages 1–6, New York, NY, USA. ACM.

[Gärtner 1999] Gärtner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput. Surv.*, 31(1):1–26.

[Hoffmann et al. 2007] Hoffmann, G., Trivedi, K., e Malek, M. (2007). A best practice guide to resource forecasting for computing systems. *Reliability, IEEE Transactions on*, 56(4):615–628.

[Kon e Goldman 2008] Kon, F. e Goldman, A. (2008). *Grades Computacionais: Conceitos Fundamentais e Casos Concretos*, chapter 2, pages 55–104. Sociedade Brasileira de Computação, Belém do Pará.

[Li et al. 2002] Li, L., Vaidyanathan, K., e Trivedi, K. S. (2002). An approach for estimation of software aging in a web server. In *ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*, page 91, Washington, DC, USA. IEEE Computer Society.

[Lorena e Carvalho 2007] Lorena, A. C. e Carvalho, A. C. P. L. F. (2007). Uma introdução às support vector machines. *Revista de Informática Teórica e Aplicada*, XIX:43–67.

[Tanenbaum e Steen 2007] Tanenbaum, A. S. e Steen, M. V. (2007). *Sistemas Distribuídos Princípios e Paradigmas*. Pearson, São Paulo.

[Turnbull e Alldrin 2003] Turnbull, D. e Alldrin, N. (2003). Failure Prediction in Hardware Systems. *UCSD CSE221 Project*.

[van Renesse et al. 1998] van Renesse, R., Minsky, Y., e Hayden, M. (1998). A gossip-style failure detection service. In *Middleware*, volume 98, pages 55–70.

[Witten e Frank 2005] Witten, I. H. e Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition.

[Xue et al. 2007] Xue, Z., Dong, X., Ma, S., e Dong, W. (2007). A survey on failure prediction of large-scale server clusters. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007. SNPD 2007. Eighth ACIS International Conference on*, volume 2, pages 733–738.

# Tolerância a Falhas utilizando Protocolos de *Gateway* Redundantes

**Alessandro Kraemer[1], Kaio Vilar[2], Alfredo Goldman[3]**

[1]Coordenação de Informática – Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão – PR – Brazil

[2]Centro de Atendimento Técnico ao Cliente – Embratel - Brasília – DF - Brazil

[3]Departamento de Ciência da Computação – Instituto de Matemática e Estatística – Universidade de São Paulo (IME-USP)
São Paulo – SP - Brazil

kraemer@utfpr.edu.br, kavilar@embratel.com.br, gold@ime.usp.br

***Abstract.*** *The goal of this paper is to compare fault-tolerant protocols inside of redundancy gateway context. For this purpose, performance tests were conducted with specific protocols, like HSRP, VRRP and GLBP. The results indicate which protocol will recover more quickly from the interruption of the default gateway.*

***Resumo.*** *O objetivo deste artigo é comparar protocolos de tolerância a falhas em contexto de gateway redundante. Para tanto, foram executados testes com protocolos específicos, como HSRP, VRRP e GLBP. Os resultados apontam qual protocolo se recupera mais rapidamente a partir da interrupção do gateway padrão.*

## 1. Introdução

A tecnologia de tolerância a falhas é considerada um requisito em sistemas de missão crítica [Song e Choi, 2001, p.315]. A redundância é uma técnica importante deste contexto. Ela ocorre em vários níveis, desde a comunicação envolvendo o cabeamento até a replicação de servidores e ativos de rede. A duplicação de *gateways* também é um exemplo de uso de redundância [Lefrève et al., 2006, p.258].

Shinn [2009] descreve mecanismos de *gateways* redundantes. A característica principal desses protocolos é que vários *gateways* trocam mensagens a fim de perceber se podem assumir o lugar de outro, seja por falha ou por necessidade de balanceamento de carga.

Este artigo apresenta os mecanismos de redundância utilizados pelos protocolos HSRP (*Hot Standby Router Protocol*), VRRP (*Virtual Router Redundancy Protocol*) e GLBP (*Gateway Load Balance Protocol*). Por fim, é apresentado um cenário de rede com *gateways* redundantes utilizado para avaliar esses protocolos. O cenário foi implementado em laboratório de redes com roteadores CISCO e computadores com Linux, procurando gerar estresse de acessos HTTP, como comumente ocorre em situações reais de uso da Internet.

## 2. Protocolos de *Gateway* Redundantes

### 2.1. Protocolo HSRP

O HSRP é padronizado pela RFC 2281 e consiste em configurar grupos de roteadores com endereçamento IP e MAC distintos. Assim, existirão vários *gateways* para uma mesma rede. Os roteadores possuem um segundo endereço, chamado de IP Virtual e MAC Virtual. Este endereço é idêntico em todos os roteadores do domínio [Shinn, 2009, pp.274]. Contudo, os computadores da rede conhecem apenas um *gateway* virtual.

Os roteadores envolvidos neste processo trocam mensagens conhecidas como *Hello,* a fim de verificar o estado operacional do seu vizinho. Dentro do grupo de roteadores, apenas um roteador fica em estado Ativo e os demais ficam em estado de *backup*. Embora o HSRP não possua balanceamento de carga nativo, é possível implementá-lo criando vários grupos com IP Virtuais distintos [Shinn, 2009, pp.274].

### 2.2. Protocolo VRRP

O VRRP é uma alternativa ao HSRP, detalhado pela RFC 3768. Neste protocolo, os endereçamentos reais e virtuais podem participar efetivamente do mecanismo de redundância. A comunicação entre os *gateways* ocorre através de mensagens similares ao *Hello,* mas são conhecidas como *Link-State Advertisement* (LSA). O papel de enviar essas mensagens é do roteador conhecido como Mestre [Shinn 2009, p.274] [Song e Choi, 2001, p.317]. Quando o Mestre falha, outro roteador com prioridade logo abaixo percebe a ausência de LSA e assume seu papel.

No VRRP ocorre balanceamento de carga. O tráfego é balanceado porque diferentes endereços de *gateway* são distribuídos entre as estações cliente.

### 2.3. Protocolo GLBP

A principal diferença do GLBP em relação aos demais é que ele consegue atribuir diferentes endereços MAC para um mesmo IP Virtual. No HSRP e no VRRP existe um *gateway* principal e os outros são *backup*, enquanto no GLBP, os *gateways* de *backup* são também conhecidos como encaminhadores. Os encaminhadores também são ativos.

No GLBP existem dois tipos de *gateways* ativos: o *Gateway* Virtual Ativo (AVG) e o *Gateway* Virtual Encaminhador (AVF). O AVG é eleito pelo grupo e os AVF são seus *backups.* A cada solicitação ARP feita ao AVG é devolvido o MAC Virtual de outro roteador AVF. Com este mecanismo, o endereço MAC do *gateway* armazenado na tabela ARP do cliente não é o mesmo em todas as estações, permitindo o balanceando da carga [Satapati et al., 2004, pp.02].

### 2.4. Comparação entre os Protocolos

O objetivo da comparação é descobrir quanto tempo é gasto até que os elementos de rede percebam que o *gateway* principal foi interrompido e que um novo caminho deve ser seguido. Embora este critério seja importante, ele não pode ser analisado isoladamente. Também é importante considerar os padrões de tempo de cada protocolo. Quando um roteador recebe uma mensagem *Hello/LSA,* ele a torna válida somente por um período de tempo, isto é conhecido como *holdtime*. Se o *holdtime* exceder e não

chegar outra mensagem, o enlace é considerado falho, dando inicio a substituição de *gateway*. O ajuste desses parâmetros pode alterar o resultado da comparação. Neste artigo, os protocolos são avaliados considerando seus padrões de tempo (Tabela 1).

**Tabela 1. Características dos protocolos de *gateway* redundantes.**

| Protocolo | Equipamentos necessários | Forma de Endereçamento | Balanceamento de carga | Tempo padrão |
|---|---|---|---|---|
| HSRP proprietário | Roteadores CISCO | Um IP e um MAC Virtuais idênticos em cada roteador do grupo | Não é nativo | 3 seg. (*Hello*)<br>10 seg. *(Holdtime)* |
| VRRP solução aberta | Roteadores dedicados ou Plataformas Linux | Um ou mais IP e MAC Virtuais. Os IP e MAC reais também podem ser utilizados | Cada estação cliente recebe um endereço de gateway diferente | 1 seg. (*LSA)*<br>3 seg. *(Holdtime)* |
| GLBP proprietário | Roteadores CISCO | Um IP Virtual e vários MAC virtuais que identificam os roteadores do grupo | Cada estação cliente pode receber MAC distinto a cada solicitação ARP | 3 seg. (*Hello)*<br>10 seg. *(Holdtime)* |

A Figura 1 representa o cenário elaborado para avaliação destes protocolos. Nesse cenário, cada computador tem o papel de executar acessos HTTP por meio de um aplicativo Java elaborado para este fim. O tempo total do acesso considera desde a conexão até o recebimento do conteúdo do site. O teste consiste em interromper o *gateway* padrão enquanto são realizados vários acessos ao servidor. Nesses momentos, os aplicativos mantêm a sessão com o servidor, mas o tempo gasto se diferencia do acesso normal. O mesmo sistema de avaliação se repetiu 10 vezes para cada protocolo.


**Figura 1. Cenário para avaliação dos protocolos de *gateway*.**

A Figura 2 apresenta o resultado da avaliação, indicando o tempo gasto para que a sessão do momento da interrupção fosse direcionada ao novo *gateway* padrão. Por exemplo, durante a segunda interrupção, tanto para o GLBP quanto para o HSRP, o tempo da sessão HTTP foi o mesmo nas duas estações (aproximadamente 13 segundos). Por outro lado, no VRRP foram gastos apenas 3 segundos.

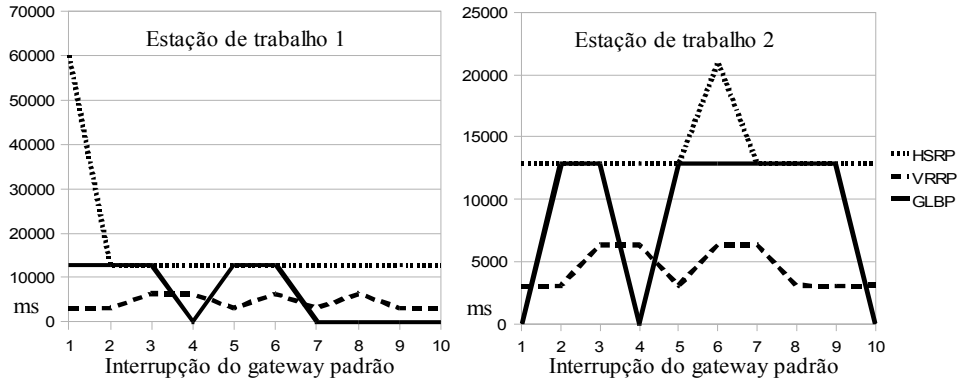



**Figura 2. Resultado da avaliação.**

Embora os *gateways* tenham sido interrompidos apenas 10 vezes em cada configuração de protocolo, foi possível observar um comportamento padrão. O VRRP consegue ser o mais rápido na definição do novo *gateway* padrão, chegando algumas vezes a consumir apenas ¼ do tempo dos demais protocolos, e no pior caso chegando a 50%. O GLBP mantém um tempo linear de recuperação e em alguns casos consegue ser totalmente transparente, com tempos muito próximos ao estado anterior da interrupção do *gateway*, como apresenta a Figura 2 ao declínio linear do 4º processo de interrupção. Por fim, o HSRP possui tempos de recuperação similares ao GLBP, mas sem os casos especiais de acesso com tempos imperceptíveis. A desvantagem do HSRP é que em seu estado de configuração padrão não existe balanceamento de carga, o que pode ter influenciado no resultado final.

## 3. Considerações Finais

A principal característica dos protocolos HSRP, VRRP e GLBP é que eles gerenciam um mesmo IP de *gateway* virtual, distribuído entre um grupo de roteadores com endereços IP reais diferentes. O administrador de rede pode ajustar os padrões de tempo das mensagens de estado, tentando melhorar o desempenho do protocolo. Para as estações de trabalho, todo esse processo é transparente, pois elas conhecem apenas o IP Virtual, que continua o mesmo, independentemente do *gateway* que foi interrompido.

Entre os protocolos explorados, o VRRP demonstrou ser o que se recupera/elege mais rapidamente o novo gateway padrão. Uma razão para isto é que seus padrões de tempo de mensagens *Hello* e *holdtime* são menores que os demais protocolos. Outra característica importante é o custo da implantação. Os três protocolos explorados podem ser configurados em roteadores CISCO, mas o VRRP também está disponível em plataformas Linux, o que também pode ser considerada uma vantagem em relação aos demais. O protocolo GLBP tem melhor desempenho que o HSRP, já que os padrões de tempo são os mesmos e em alguns acessos o GLBP demonstrou ser totalmente transparente. Demais testes devem ser realizados ajustando os tempos das mensagens de estado dos protocolos GLBP e VRRP.

## Referências


Lefèvre, L., Neira, P. and Gasca, R. M. (2006). High Availability support for the design of stateful networking equipments. In IEEE Computer Society, pages 254-261. Proceedings of the First International Conference on Availability, Reability and Security.

Shinn, S. K. (2009). Fault Tolerance Virtual Router for Linux Virtual Server. In IEEE Computer Society, pages 273-275. International Conference on Future Networks.

Satapati, S., Wilson, I. H. and McLaggan, D. Network Address Translation with Gateway Load Distribution. US Patent 2004/0215752, filed Mar, 28, 2003, and issued Oct, 28, 2004.

Song, S. and Choi, B. (2001). Scalable Fault-Tolerant Network Design for Ethernetbased Wide Area Process Control Network Systems. In IEEE Computer Society, pages 315-323. 8th IEEE International Conference on Emerging Technologies and Factory Automation.

# Tolerância a falhas no armazenamento distribuído de dados em grades oportunistas

**Pablo Laura-Huaman**[1], **Raphael Y. de Camargo**[2]

[1]Instituto de Matemática e Estatística
Universidade de São Paulo (IME-USP) – São Paulo, SP – Brazil

[2]Centro de Matemática, Computação e Cognição
Universidade Federal do ABC (UFABC) – São Paulo, SP – Brazil

pablolh@ime.usp.br, raphael.camargo@ufabc.edu.br

***Resumo.*** *OppStore é um middleware que implementa operações de armazenamento e recuperação de dados utilizando espaço em disco ocioso das máquinas de uma grade oportunista. Os arquivos são codificados em fragmentos redundantes, que são distribuídos em diferentes máquinas, e podem ser reconstruídos utilizando apenas um subconjunto destes fragmentos. Mas as máquinas componentes da grade podem falhar, ficar inacessíveis ou passar de ociosas para ocupadas inesperadamente, impedindo o acesso aos fragmentos nelas armazenados. Neste trabalho, definimos, implementamos e avaliamos mecanismos de tolerância a falhas que recuperam os fragmentos perdidos devido à falhas ou indisponibilidades nas máquinas da grade.*

## 1. Introdução

Grades oportunistas são compostas por máquinas compartilhadas, que tipicamente possuem quantidades significativas de espaço livre em disco, que poderia ser utilizado durante os períodos de ociosidade das máquinas. OppStore [de Camargo and Kon 2007] é um sistema de middleware que realiza o gerenciamento das máquinas de uma grade computacional oportunista, permitindo o armazenamento de dados utilizando o espaço em disco livre durante seu período de ociosidade. Para permitir uma maior disponibilidade dos dados armazenados, OppStore codifica os arquivos em fragmentos redundantes, utilizando uma versão otimizada do algoritmo de dispersão de informação (IDA)[Rabin 1989]. Os fragmentos gerados são então distribuídos em diferentes máquinas da grade. Mas as máquinas da grade podem falhar, ficar inacessíveis ou passar de ociosas para ocupadas inesperadamente, impedindo o acesso aos fragmentos nelas armazenados. Um mecanismo de tolerância a falhas que permita manter a disponibilidade destes fragmentos é um quesito importante para este sistema.

Neste trabalho, definimos, analisamos, implementamos e avaliamos dois mecanismos de tolerância a falhas que permitem a recuperação de fragmentos perdidos devido à falhas ou indisponibilidades nas máquinas da grade. O primeiro mecanismo realiza a reconstrução do arquivo original, que é utilizado para gerar novamente os fragmentos perdidos. O segundo mecanismo mantém uma cópia adicional de cada fragmento, que é utilizada para recuperar os fragmentos perdidos sem a necessidade de reconstruir o arquivo original. Por meio de simulações, avaliamos o custo de cada mecanismo, como o número de mensagens geradas e a quantidade de tráfego na rede, e a capacidade de cada mecanismo de manter a disponibilidade dos arquivos armazenados na presença de falhas.

## 2. Armazenamento distribuído de dados no OppStore

As máquinas no OppStore são organizadas como uma federação de aglomerados, onde cada aglomerado é um conjunto de máquinas fisicamente próximas e pertencentes a um mesmo domínio administrativo. Estes aglomerados são conectados por uma rede peer-to-peer estruturada, usando Pastry[Rowstron and Druschel 2001] como substrato.

OppStore contém três componentes principais que são: o gerenciador de repositórios de dados do aglomerado (CDRM), o repositório autônomo de dados (ADR) e o intermediador de acesso (*access broker*). Cada aglomerado possui uma máquina de gerenciamento do aglomerado, que instancia o CDRM. As demais máquinas do aglomerado são utilizadas como repositórios de dados e executam o módulo ADR. A Figura 1 mostra a arquitetura do OppStore.

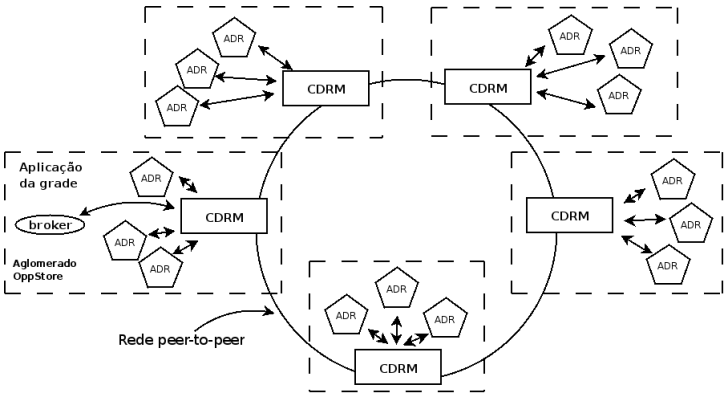



**Figura 1. Arquitetura do OppStore**

O processo de armazenamento de dados é realizado por meio dos seguintes passos:

1. O *access broker* codifica um arquivo em fragmentos redundantes e atribui um identificador para cada fragmento.
2. Após a codificação do arquivo, o *access broker* envia a lista de fragmentos codificados ao CDRM do seu aglomerado, que cria uma mensagem para cada identificador de fragmento. O CDRM envia cada mensagem para a rede peer-to-peer, que a roteia para o CDRM pelo seu identificador.
3. Ao receber a mensagem de armazenamento, cada CDRM seleciona um ADR do seu aglomerado e devolve o endereço de rede do ADR ao CDRM que enviou a mensagem. Este repassa cada endereço recebido para o *access broker*.
4. Após receber a lista de endereços, o *access broker* transfere o conteúdo de cada fragmento redundante ao seu respectivo ADRs.
5. O *access broker* notifica o CDRM do seu aglomerado, que cria uma estrutura chamada *file fragment index* (FFI) que contém a localização e o identificador de cada fragmento. O CDRM cria uma mensagem contendo o FFI criado a envia para o armazenamento remoto no CDRM responsável pelo seu identificador.

## 3. Mecanismos de recuperação de fragmentos

Implementamos dois mecanismo de recuperação de fragmentos perdidos, que são responsáveis por reconstruir fragmentos perdidos devido à saída de máquinas da grade, mantendo um número de fragmentos suficiente para reconstruir os arquivos armazenados.

**Mecanismo 1: Reconstrução do arquivo armazenado.** Este mecanismo recupera os fragmentos através da reconstrução do arquivo original, que é utilizado para gerar novamente os fragmentos que foram perdidos. O processo de reconstrução de fragmentos perdidos é o seguinte:

1. O CDRM de cada aglomerado monitora suas máquinas. Sempre que ele detecta que um ADR se torna inacessível, ele verifica quais fragmentos estavam armazenados no ADR perdido e, para cada fragmento que estava armazenado no ADR, e notifica o CDRM responsável pelo FFI do arquivo contendo o fragmento. Esta mensagem é então roteada pela rede peer-to-peer.
2. Ao receber uma mensagem de perda de fragmento, os CDRMs atualizam sua lista de identificadores de fragmentos que estão armazenadas no FFI, marcando o fragmento como indisponível. Se o número de fragmentos disponível no FFI fica abaixo de um limiar $L$ pré-estabelecido, o mecanismo de recuperação é iniciado. O limiar evita que o mecanismo de recuperação seja iniciado cada vez que um fragmento é perdido.
3. O CDRM responsável por iniciar o mecanismo envia ao *access broker* o FFI contendo a lista de fragmentos disponíveis. O *access broker* reconstrói o arquivo original e gera novamente os fragmentos que foram perdidos.
4. Para cada fragmento recuperado, o *access broker* solicita ao um novo endereço para armazenamento, transfere os fragmentos direitamente para os ADRs e notifica o CDRM sobre os novos endereços. Este último envia uma mensagem para o CDRM que armazena o FFI do arquivo com os novos endereços dos fragmentos.

**Mecanismo 2: Cópia local dos fragmentos.** Como alternativa para manter a disponibilidade dos fragmentos devido à saída ou indisponibilidade de computadores da grade, introduzimos uma estratégia de replicação, que consiste em manter uma cópia adicional de cada fragmento armazenado. Esta cópia é colocada em uma máquina do mesmo aglomerado do fragmento original. Sempre que um fragmento é perdido, o CDRM do aglomerado gera um novo fragmento a partir da cópia armazenada. Os passos deste mecanismo são:

1. Quando detecta a saída de um computador de seu aglomerado, o CDRM localiza a cópia de cada fragmento que estava naquela máquina. Para tal, o CDRM mantém uma tabela contendo a localização dos pares de cópia de cada fragmento.
2. O CDRM então armazena uma nova cópia do fragmento em outra máquina do mesmo aglomerado e atualiza sua tabela.
3. O CDRM envia uma mensagem com o novo endereço do fragmento para o CDRM responsável pelo FFI procurado. Ao receber a mensagem, este CDRM atualiza o FFI do arquivo com o novo endereço.

## 4. Resultados experimentais

Para avaliar o mecanismo, realizamos a simulação de uma grade contendo 30 aglomerados com 50 máquinas em cada. Armazenamos 900 arquivos e simulamos um período de 1 mês e durante este período, 750 máquinas falharam. Realizamos experimentos utilizando os 2 mecanismos, com o intuito de avaliar a sobrecarga dos mecanismos.

A Tabela 1 mostra o número de mensagens geradas pelo primeiro mecanismo, onde cada arquivo foi codificado em 6 fragmentos, dos quais 3 eram suficientes para sua recuperação. Executamos o protocolo de reconstrução com 3 diferentes limiares $L$ (3, 4 e 5). A Tabela 2 mostra o número de mensagens geradas para o mecanismo 2, nos casos onde o arquivo é codificado em 6, 12 ou 24 fragmentos, dos quais 50% são necessários para reconstruir o arquivo original.

| Tipos de mensagens | limiar 3 | limiar 4 | limiar 5 |
|---|---|---|---|
| Mensagens de atualização FFI | 2869 | 2726 | 2511 |
| Mensagens requisitando endereço | 1686 | 1514 | 1089 |
| Mensagens de atualização final FFI | 488 | 702 | 1089 |
| Número total de mensagens | 4854 | 4952 | 4689 |
| Números de vezes de início do protocolo | 594 | 849 | 1254 |

**Tabela 1. Mensagens criadas no primeiro mecanismo**

| Tipos de mensagens | 6 fragmentos | 12 fragmentos | 24 fragmentos |
|---|---|---|---|
| Mensagens de atualização de FFIs | 7452 | 15305 | 28648 |
| Número de fragmentos movimentados | 7452 | 15305 | 28648 |

**Tabela 2. Mensagens criadas no segundo mecanismo para 6, 12 e 24 fragmentos**

O segundo mecanismo gerou uma maior quantidade de mensagens, uma vez que existe um maior número de fragmentos armazenados nos ADRs. Além disso, este mecanismo utiliza o dobro da quantidade de espaço em disco. Por outro lado, como no segundo mecanismo o arquivo original não precisa ser reconstruído, a quantidade de tráfego entre os aglomerados é menor, dado que os fragmentos não precisam ser transferidos para a reconstrução do arquivo.

Verificamos também que, para o mecanismo 1, quanto maior o valor do limiar, maior o número de vezes que o mecanismo é iniciado. Isso porque estamos antecipando as possíveis indisponibilidades de fragmentos armazenados em diferentes ADRs de acordo com o grande número de falhas. Por outro lado, isto tende a aumentar a disponibilidade dos arquivos. Para o mecanismo 2, quanto maior o número de fragmentos gerados por arquivos, maior o número de mensagens, uma vez que cada ADR irá armazenar um maior número de fragmentos.

Os resultados indicam que o mecanismo 1 é o mais adequado quando consideramos o número de mensagens geradas e o espaço em disco utilizado. Mas o segundo é melhor quando queremos minimizar a quantidade de dados transferida entre aglomerados.

## Referências


de Camargo, R. and Kon, F. (2007). Design and implementation of a middleware for data storage in opportunistic grids. In *Seventh IEEE International Symposium on Cluster Computing and the Grid, 2007. CCGRID 2007*, pages 23–30.

Rabin, M. (1989). Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348.

Rowstron, A. and Druschel, P. (2001). Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *18th IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany.

# Sessão 2

# Computação em Nuvem

# *Memory Dispatcher*: Gerência de Recursos em Ambientes Virtualizados

**Artur Baruchi, Edson Toshimi Midorikawa**

Escola Politécnica da Universidade de São Paulo (EPUSP) – Laboratório de Arquitetura de Computadores e Computação Distribuída (LAHPC)

{artur.baruchi,edson.toshimi}@poli.usp.br

***Abstract.*** *The main advantage when using virtualization is better resource usage. However, some resources, like memory, aren't trivial to be managed in a virtual environment. This work presents the Memory Dispathcer, it's a mechanism to balance memory among Virtual Machines, based on Page Faults. Preliminary results, using two benchmarks showed that memory has been balanced successfully and an acceptable performance.*

***Resumo.*** *Uma das maiores vantagens ao utilizar um ambiente virtual é a melhor utilização dos recursos computacionais. Entretanto, alguns recursos, como a memória, ainda são difíceis de serem gerenciados em um ambiente virtual. O presente trabalho apresenta o Memory Dispatcher, um mecanismo de balanceamento de memória para Máquinas Virtuais, baseado em Falta de Páginas. Resultados obtidos com dois benchmarks mostraram que a memória foi melhor balanceada e o desempenho se manteve aceitável.*

## 1. Introdução

Apesar do conceito das Máquinas Virtuais (MV) serem do final do inicio da década de 70 [Goldberg, 1974], a virtualização ganhou muita força com o advento de processadores multicores e com o barateamento de componentes como a memória. O principal objetivo da virtualização é melhorar o gerenciamento de recursos que se tornaram subutilizados. Diversas propostas de virtualização para a plataforma x86 podem ser encontradas na Internet e na literatura, algumas já consolidadas como o VMWare e Xen e outras ainda em processo de consolidação, como o KVM.

Apesar da virtualização não ser um conceito novo, a virtualização na plataforma x86 é mais recente. A virtualização na plataforma x86 é mais complexa e de acordo com a teoria de Popek e Goldberg [Smith, 2005] não é uma plataforma que suporta, de forma nativa, a virtualização. Para que este problema fosse contornado, algumas propostas, como a Paravirtualização [Whitaker, 2002], foram concebidas e mostraram-se muito eficientes.

Além de características atraentes para outras áreas da computação já consolidadas, a virtualização viabilizou antigas idéias como a utilização da computação como serviço [Parkhill, 1966]. A idéia do uso da computação como serviço deu origem ao termo computação em nuvem, que vem sendo amplamente discutida e estudada atualmente ([Kandukuri, 2009], [Vecchiola, 2009]). O papel da virtualização na computação em nuvem é tão importante que em algumas definições deste paradigma é utilizado o termo virtualização [Buyya et al, 2008]:

*"... tipo de sistema paralelo e distribuído que consiste em uma coleção de computadores **virtualizados** e interconectados que são dinamicamente fornecidos e apresentados como um ou mais recursos computacionais unificados fundamentados em um acordo de nível de serviço..."*

Neste trabalho serão apresentados alguns resultados obtidos com o uso do *Memory Dispatcher* (MD). Na sessão 2 é apresentada a arquitetura do MD, os resultados serão abordados na sessão 3 e por fim, na sessão 4, serão apresentadas as conclusões.

## 2. Arquitetura do *Memory Dispatcher*

Para a implementação do MD foi utilizado o Monitor de Máquinas Virtuais (MMV) Xen [Barhan et al., 2003]. Toda a implementação do MD foi realizada em linguagem C. A comunicação entre as MVs foi implementada através de memória compartilhada, chamada de Xenstore [Chisnnal, 2007].

O MD utiliza uma arquitetura análoga à arquitetura cliente-servidor. Na MV foi implementado um processo (*daemon*) que coleta as informações da MV e realiza o cálculo da quantidade de memória estimada para suprir as necessidades da MV. Esta informação é então passada ao Dom0 que verifica a possibilidade de alocar a quantidade de memória para a MV.

Durante a implementação, o cálculo da memória era realizado pelo Dom0 (MV com prioridade administrativa no Xen), por meio de informações passadas pela MV. Entretanto, esta abordagem apresentou dois problemas:

- Excesso de dados a serem escritos/lidos da região de memória compartilhada;
- Sobrecarga de processamento para o Dom0; com duas Mvs, o processamento a sobrecarga é aceitável, mas para um número maior de MVs pode tornar-se inviável;

A parte do MD que fica em execução na MV, ao iniciar, indica por uma *flag* no Xenstore de que aquela MV está sob gerenciamento do MD. Quando o processo termina, esta *flag* é removida do Xenstore e o MD retira esta MV das suas estruturas internas de gerenciamento. Com esta característica, é possível que em um mesmo ambiente físico haja MVs gerenciadas e não gerenciadas pelo MD (por exemplo, pode ser necessário que uma MV possua uma quantidade fixa de memória).

## 3. Resultados

A avaliação do mecanismo foi realizado com dois *benchmarks*. O primeiro *benchmark* utilizado foi o DBench [DBench], um *benchmark* que simula um servidor Samba e de comportamento I/O *Bound*. O DBench permite que a quantidade de usuários acessando o servidor seja configurado e com isso aumentar ou diminuir a carga de trabalho no ambiente. O segundo *benchmark* utilizado foi a compilação do Kernel do Linux [Kernel]. A compilação do Kernel simula uma aplicação CPU Bound.

Os testes foram repetidos por 15 vezes e como parametro de comparação foram feitos testes com os mesmos *benchmarks* utilizando MVs com memória estática. Após a realização das 15 repetições (por conta de limitações do espaço nem todos os resultados

serão apresentados), foi calculado a média e a distribuição *student*-t [Pearson, 1939] com 95% de confiabilidade. Ao todo foram executados 600 testes (120 testes de compilação do *Kernel* e 480 com o DBench). O DBench foi executado com 50, 75, 100 e 150 clientes simulando acesso ao servidor Samba.

A execução dos testes foi realizada em duas MVs (MV01 e MV02). Os benchmarks foram executados simultaneamente na MV01 e MV02. Para isso, foram criados scripts de automação dos testes, com o objetivo de manter o sincronismo. A execução de forma concorrente é importante, pois raramente um ambiente físico conterá apenas uma única MV.

O desempenho apresentado na MV01 manteve-se satisfatório para o DBench (Figura 1), ficando próximo das configurações fixas de memória de 2,6GB (quanto maior a barra do gráfico melhor). Para o segundo benchmark (Figura 2) o MD também apresentou um desempenho médio aceitável quando comparado com as configurações de memória fixa.

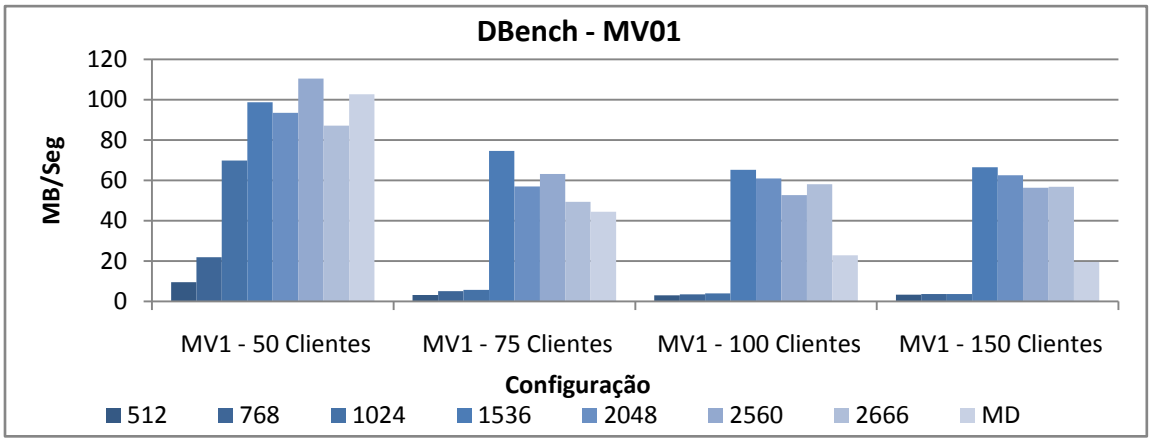



**Figura 1. Gráfico com o Desempenho do DBench na MV01.**

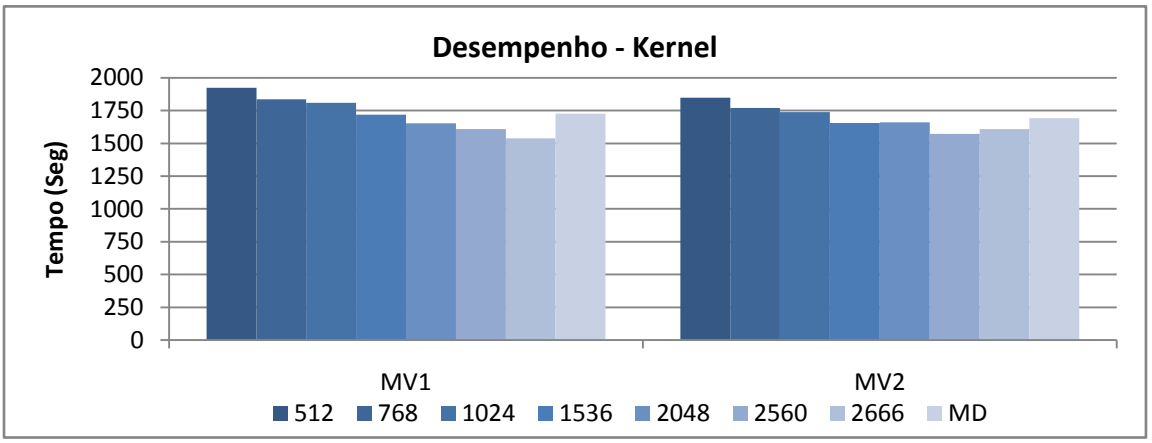



**Figura 2. Gráfico com o Desempenho da Compilação do *Kernel*.**

## 4. Conclusões

Este trabalho apresentou o *Memory Dispatcher*, cujo objetivo é melhorar o gerenciamento de memória em um ambiente virtualizado. O MD foi implementado no Xen e para sua avaliação foram executados dois *benchmarks* com comportamentos distintos. Através dos resultados obtidos, foi possível observar que o desempenho das MVs permaneceu aceitável em comparação com as configurações fixas.

O principal fator observado para perda de desempenho é o constante acionamento do *Balloon Driver* [Waldspurger, 2002] nas MVs. Este mecanismo força o Sistema Operacional (SO) da MV a realizar paginação e as paginas que são descartadas pelo SO são então devolvidas ao Xen, que por sua vez poderá alocar estas páginas para outras MVs. Desta forma, ao acionar o *Balloon Driver* o SO será sobrecarregado constantemente para buscar páginas descartáveis e como conseqüência perder desempenho.

Outros critérios de adição e remoção de memória estão sendo avaliados, como o uso de Média Móvel Exponencial e algoritmos de Inteligência Artificial. Entretanto, o maior desafio ainda a ser transposto é criação de um mecanismo que não cause a sobrecarga observada pelo *Balloon Driver*.

## Referências


Barham, P., B. Dragovic, et al. (2003). "Xen and the Art of Virtualization". 19th ACM Symposium on Operationg System Principles. Bolton Landing, NY, USA: ACM Press. p. 164-177.

Buyya, R. et al. (2008) "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities". In Proc. 10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008), IEEE CS Press, Sept. 25–27.

Chisnnal, David. (2007) "The Definitive Guide to the Xen Hypervisor". 1a Edição. Ed. Prentice Hall PTR.

DBench. http://dbench.samba.org/

Kandururi, Balachandra; Paturi, Ramakrishna; Rakshit, Atanu. (2009) "Cloud Security Issues". IEEE International Conference on Services Computing.

Kernel. http://kernel.org/

Pearson, E. S. (1939) "Student as a Statistian". Biometrika, v. 20. p. 210-250.

Smith, James E.; NAIN, Ravi. Virtual. (2005) " Machines: Versatile Platforms For Systems And Processes" Ed. Morgan Kaufmann, 1a Edição.

Vecchiola, Christian; Pandey, Suraj; Buyya, Rajkumar. (2009). "High-Performance Cloud Computing: A view of Scientific Applications". Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks.

Waldspurger, C. A. (2002). "Memory Resource Management in VMWare ESX Server". Proceeding of the 5th Symposium on Operating System Desing and Implementation. Boston, MA. p. 181-194.

Whitaker, A.; Shaw M.; Gribble, S. D. (2002) "Denali: Lightweight Virtual Machines for Distributed and Networked Applications". Technical Report 02-02-01, University of Washington.

# Arquitetura de Plataforma para Computação em Nuvem visando a Interoperabilidade

**Charles B. Rodamilans[1], Edson T. Midorikawa[1]**

[1]Departamento de Engenharia de Computação e Sistemas Digitais
Universidade de São Paulo – São Paulo, SP – Brasil

{charles.rodamilans,edson.midorikawa}@poli.usp.br

***Abstract.*** *The lack interoperability between different Clouds, in the level of infrastructure or platform as a service, has restricted the adoption of companies because of the fear of leaving their business to a single service provider. The aim of this paper is to propose an platform architecture in Cloud Computing focusing an interoperability.*

***Resumo.*** *A falta interoperabilidade entre diferentes Nuvens, seja no nível de infraestrutura ou de plataforma como serviço, tem restringindo a adoção de empresas, em razão do receio de atrelar seu negócio a um único provedor de serviço. O objetivo deste trabalho é propor uma arquitetura de Plataforma em Computação em Nuvem visando à interoperabilidade.*

## 1. Computação em Nuvem

Existem várias definições para Computação em Nuvem (*Cloud Computing*). Uma delas é que Computação em Nuvem refere-se tanto às aplicações entregues com serviços utilizando a Internet quanto ao hardware e sistema software dos Centro de Dados de onde provêm esses serviços também se referem a Computação em Nuvem [1].

Computação em Nuvem assemelha-se aos serviços de utilidade pública (*Utility Computing*), como fornecimento de energia elétrica, água. No modelo tradicional, paga-se o aluguel de hospedagem e/ou licença do software. O cliente aluga uma máquina, um serviço de armazenamento de dados, compra a licença vitálicia ou temporária de uma versão do software. O valor é pago independentemente da real utilização dos recursos e dos softwares. No modelo de Computação em Nuvem, paga-se pelo serviço que foi utilizado, seja no nível de hardware (Infraestructure as a Service - IaaS), plataforma (Platform as a Service - PaaS) ou de software (Software as a Service - SaaS). O modelo de Utilidade Pública para computação não é um conceito novo [1]. Devido às limitações tecnológicas da época, apenas agora foi viabilizado.

Como exemplo de serviço de hardware (IaaS), a Amazon EC2 [2] fornece um serviço (ambiente) de Computação Virtual. O usuário deste serviço paga pelo a quantidade de tráfego de rede, de uso da CPU, memória e pelo tipo pelo sistema operacional que utilizou. No nível de software (SaaS), a Microsoft está oferecendo o pacote Office em Nuvem. O usuário pode acessar a ferramenta Word pelo navegador e pagar pelo tempo que realmente utilizou a ferramenta.

## 2. Problemas relacionados a Computação em Nuvem

Atualmente, os serviços de plataformas são oferecidos para a utilização de uma Nuvem específica e com limitação de linguagem. A utilização dessas plataformas causa o aprisionamento e o cliente fica vulnerável às políticas da Nuvem. Esse aprisionamento ocorre em decorrência da dificuldade de transição das aplicações de maneira transparente e da falta de interoperabilidade entre as Nuvens. Como exemplo, o AppEngine (Google), Azure (Microsoft) [1], VMsales (Vmware+SalesForce) oferecem uma plataforma de desenvolvimento de aplicações proprietárias, específica para suas

infraestruturas. Aneka [4,5] é um software de plataforma para o desenvolvimento de aplicações em Nuvem; este é restrito a aplicações .NET e é proprietário.

Uma forma de contornar este problema é a criação de uma plataforma aberta, que interligue as diferentes Nuvens e permita a migração da aplicação do usuário. Por conta dos serviços de plataforma (PaaS) disponíveis serem fechados, essa interligação fica restrita ao nível de infraestrutura (IaaS).

A criação de uma Máquina Virtual (Virtual Machine, VM) no nível de Infraestrutura da Nuvem é apresentada na Figura 1. O usuário acessa e gerencia os serviços disponibilizados pela Infraestrutura através de um Interface de Nuvem (**1**). O Open Cloud Computing Interface (OCCI) é uma proposta de padronização de interface de programação para a Nuvem baseado em diversas interfaces de IaaS. A Interface da Nuvem se comunica com o Software de Infraestrutura (**2**), sendo este responsável por criar, configurar e gerenciar a infraestrutura da Nuvem (**3**) [2,3] podendo conter um ou mais gerenciador de VMs, tais como Xen, KVM, Vmware, VirtualBox [2,3]. As Nuvens públicas possuem seu próprio software de Infraestrutura, mas existem alternativas abertas, como OpenNebula, Nimbus e Eucalyptus [2,3]. A Camada de Virtualização (CV) é composta pelo gerenciador de VMs. É este, por criar, suspender, reiniciar e finalizar as VMS a partir de uma Imagem. Ao receber um pedido do Software de Infraestutura (**3**), a Camada de Virtualização cria uma VM (**5**), utilizando uma determinada Imagem (**4**).

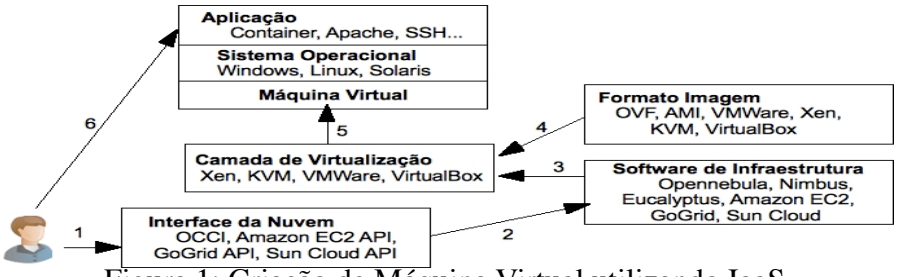


Figura 1: Criação de Máquina Virtual utilizando IaaS.

A Imagem é um arquivo que contém informações sobre a máquina virtual, tais como Sistema Operacional e Aplicações instaladas. A partir da Imagem, o gerenciador de VM pode criar uma ou mais VMs. Cada Gerenciador de VM utiliza seu formato de Imagem. Open Virtualization Format (OVF) [3] é uma especificação para uma padronização das Imagens. Após a criação da VM, o usuário pode acessar **(6)** as aplicações e gerenciar o Sistema Operacional da Máquina Virtual.

## 3. Arquitetura Proposta para Interoperabilidade

A construção de uma plataforma visando a interoperabilidade pode ser realizada de duas maneiras: migração da máquina virtual ou por *container*. A Imagem é composta pelo sistema operacional e pelas aplicações. Cada Software de Virtualização possui seu próprio formato de Imagem. Existe também a Imagem da Máquina Virtual, esta contém o SO, aplicações e estado da VM. O problema de migração de VMs está relacionado aos diferentes formatos de Imagens de VMs utilizados por cada Software de Virtualização.

Outra solução é a utilização de *containers*, ou seja, um conjunto de serviços. Esses serviços são monitoramento, execução, escalonamento. O C*ontainer* é instalado em cada VM, fornecendo subsídios para que a aplicação seja executada e possibilitando a migração da aplicação de maneira transparente aos usuários.

A seleção da Nuvem envolve uma avaliação dos serviços providos. Avalia-se a Qualidade de Serviço (QoS), tais como disponibilidade, confiabilidade, performance. O monitoramento dos recursos auxilia na tomada de decisões para o balanceamento de cargas e para a criação de mais VMs. O usuário pode fornecer informações sobre as regras do seu próprio negócio. Como exemplo, depois dos feriados as pessoas

costumam publicar suas fotos nos álbuns online. Uma empresa deste ramo pode informar ao escalonador que haverá um aumento no acesso em seu site. Assim, o escalonador pode selecionar outra Nuvem que forneça um melhor preço para uma maior quantidade de acessos. O escalonador pode separar aplicações e Banco de Dados. Aplicações podem ficar em uma Nuvem e os dados distribuídos em diferentes Nuvens, evitando o aprisionamento e problemas com leis do país no qual a Nuvem está instalada.

O objetivo deste trabalho é apresentar a pesquisa em andamento para o desenvolvimento de uma plataforma de Computação em Nuvem que permita a interoperabilidade entre diferentes Infraestruturas (IaaS), permitindo a migração da aplicação de maneira transparente ao usuário. O escalonamento é um fator fundamental para a interoperabilidade, por escolher a Nuvem que atende aos requisitos do usuário e selecionar o serviço com melhor custo.

Uma boa alternativa para a interoperabilidade que permite uma fácil migração da aplicação é a utilização de *containers*. A arquitetura proposta consiste em (a) Ferramenta de Desenvolvimento; (b) Escalonador e (c) *Containers*, conforme Figura 2.

O usuário utiliza a Ferramenta de Desenvolvimento (FD) para a criação de suas aplicações. FD fornece suporte a diferentes modelos de programação, tais como programação sequencial e paralela. O FD é composto por compiladores, aplicações e bibliotecas permitindo ao usuário desenvolver aplicação em seu próprio computador e depois enviar para a Nuvem. Possui uma interface para comunicar-se com o Escalonador e um programa para a transferir a aplicação (ex.: cliente FTP).

O usuário já deve ter contas nas Nuvens e informar ao escalonador os dados para acessá-la, ex.: certificado digital. O escalonamento é baseado no QoS, tais como disponibilidade, confiabilidade, performance e regras de negócios. O Executor é responsável por solicitar a criação das VMs. O Escalonador pode ser executado em uma máquina dedicada com acesso a Internet ou em uma VM de uma Nuvem. Escalonador possui interfaces para se comunicar com FD, *Container*, e com diversas Nuvens Públicas. Implementa funções para, criar, suspender, reiniciar e destruir VMs. Pode escalonar VMs entre diferentes Nuvens que possuam o mesmo gerenciador de VMs.

*Container* é um conjunto de serviços que fica hospedado nas VMs da Nuvem para executar e monitorar a VM e aplicação. As transferências dos dados é realizada por uma aplicação, ex.: servidor ftp. Executor é responsável por executar o código transferido pela FD. Monitoramento envia informações de memória, CPU e rede para o Escalonador através da Interface Escalonador.

A execução de uma aplicação do usuário segue os seguintes passos, conforme

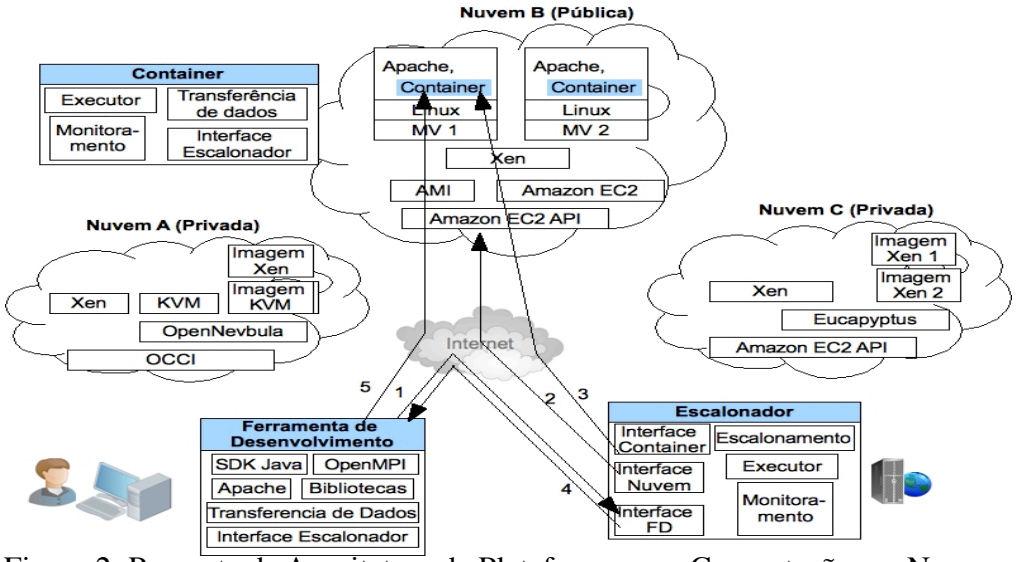


Figura 2: Proposta de Arquitetura de Plataforma para Computação em Nuvem.

apresentado na Figura 2: (**1**) depois de compilada a aplicação, a FD solicita ao Escalonador qual Nuvem e máquina virtual a ser utilizada; (**2**) Escalonador aplica o escalonamento, analisando os preços de cada Nuvem, o QoS, e seleciona qual o Nuvem a ser utilizado; com o intuito de aumentar a interoperabilidade, alguns compiladores e aplicações podem ser instalado na VM utilizando o *Container*. Executor prepara o ambiente solicitado pelo FD, utilizando a interface da Nuvem (ex.: Amazone EC2 API), cria as VMs a partir de uma Imagem (ex.: AMI) e instala o *Container*; (**3**) Escalonador se comunica com o *Container* através da Interface *Container* e inicializa o Monitoramento; (**4**) Escalonador informa a FD a Nuvem escolhida, e as informações necessárias para iniciar a Transferência de Dados; (**5**) FD faz a Transferência dos Dados, que inclui o programa da aplicação e os dados relacionados, para o *container*. Ao terminar a transferência, o *Container* executa a aplicação.

Após o inicio da execução, o Escalonador continua monitorando o *container* afim de averiguar a Qualidade de Serviço prestada. Novas informações podem ser adicionadas ao Escalonador, por exemplo modificação de preços das Nuvens, acréscimo ou remoção de novas regras de negócio via usuário. Com base nessas informações, realiza-se novamente um escalonamento para a aplicação. Nessa fase de re-escalonamento leva-se em consideração a viabilidade de migração do código da aplicação ou da VM . Para a migração do código da aplicação, é necessário observar a compatibilidade da aplicação com a versão do Sistema Operacional. Para a migração da VM, é necessário que a Nuvem execute o mesmo formato de Imagem da VM. Em ambos os tipos de migração, o tempo suspender/migrar/reiniciar a aplicação ou a VM é levado em consideração pelo escalonamento.

## 4. Discussão

Este trabalho apresenta problema de Interoperabilidade entre os provedores de IaaS, mostrando a Interface, Formato de Imagem e Imagem de Máquina Virtual. A utilização de *container* nas máquinas virtuais da Nuvem é uma alternativa para este problema. A arquitetura de plataforma proposta para Nuvem deixa transparente para o desenvolvedor o gerenciamento das Máquinas Virtuais e a migração da aplicação. O escalonamento busca o melhor custo/beneficio, ou seja, o melhor preço/QoS. Esta plataforma permite o usuário acrescentar informações que auxiliarão o escalonador. A arquitetura proposta visa solucionar o problema da Interoperabilidade das Nuvens. Os trabalhos futuros são terminar a implementação do arquitetura e avaliar o desempenho do escalonador proposto.

## Referências


[1] Armbrust, M. et al. (2009) "Above the Clouds: A Berkeley View of Cloud Computing". In: Technical Report No. UCB/EECS-2009-28.

[2] Sotomayor, B. et al. (2009) "Virtual Infrastructure Management in Private and Hybrid Clouds". In *IEEE Internet Computing*, vol. 13, ed. 5, pp. 14-22. Published by The IEEE Computer Society.

[3] Keahey, K. et al. "Sky Computing" (2009). In In *IEEE Internet Computing*, vol. 13, ed. 5, pp. 43-51. Published by The IEEE Computer Society.

[4] Vecchiola, C. et al. (2009) "High-performance Cloud Computing: A review of Scientific Applications". In Proceedings of the 10th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN 2009, IEEE CS Press, USA), Kaohsiung,Taiwan.

[5] Buyya, R. et al. (2009) "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering computing as the 5th Utility". In Future Generation Computer Systems, vol. 25, ed. 6, pp. 559-616. Published by Elsevier Science.

# Caracterização de aplicações científicas e transacionais em ambientes Cloud Computing

**Denis R. Ogura, Edson T. Midorikawa**

Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
05508-900 – São Paulo – SP – Brasil
{denis.ogura,edson.midorikawa}@poli.usp.br

***Abstract.*** *Cloud Computing is definitely the hot topic, with objectives of taking advantage of data center computational resources. Virtualizing hardware and software makes the environment scalable, redundant, and lower cost. This paper intends to characterize scientific and transactional applications in IaaS, identifying the best Virtual Machines configuration for executing these applications in Cloud Computing Environments.*

***Resumo.*** *Cloud Computing é definitivamente o assunto do momento, com objetivos de aproveitar os recursos computacionais de um data center. Virtualizando hardwares e softwares torna o ambiente escalável, redundante e de menor custo. Esse trabalho tem como objetivo caracterizar aplicações científicas e transacionais em IaaS, identificando a melhor configuração de máquinas virtuais para executar essas aplicações em ambientes de Cloud Computing.*

## 1 Introdução

Atualmente, o termo *Cloud Computing* tem sido alvo de muita discussão, pois ainda é uma tecnologia inovadora e requer visualização de diferentes pontos de vista e conceituação. Muitos entendem *Cloud Computing* como virtualização de infraestrutura (IaaS – Infraestructure As A Service), outros como virtualização de plataforma (PaaS – Plataform As A Service), e variações como software virtual (SaaS – Software As A Service).

Em um ambiente de *Cloud computing* de infraestrutura é possível implementar o sistema computacional através de MV (Máquinas Virtuais), virtualizando e executando imagens de sistemas operacionais em apenas um hardware. Dependendo de como as MV forem criadas e configuradas, os processos entrarão em execução na imagem do sistema operacional e pode haver um desempenho ruim pela falta de recursos alocados. Com uma configuração adequada dos recursos computacionais para a MV, pode-se obter melhores alternativas para um desempenho satisfatório.

As aplicações que serão executadas nas MVs têm características relevantes para serem identificadas. Nas aplicações científicas, por exemplo, tem-se a execução de diversas tarefas com um foco maior no processador. Já nas aplicações transacionais, as aplicações buscam dados e informações em diversos dispositivos de armazenamento de informações, como *storage*, HD (*Hard Disk*), SSD (*Solid State Disk*), etc.

O objetivo deste trabalho é apresentar os primeiros resultados de nossas pesquisas na identificação de características para cada tipo de aplicação em um ambiente de *cloud computing*. O escopo deste trabalho será de coletar dados de sistemas distribuídos usando o MPI (*Message Passing Interface*) para avaliar processamento em ambiente com acoplamento fraco, podendo visualizar o desempenho de processamento em máquinas virtuais diferentes. Em outra análise, será avaliado o processamento de consultas *table scan* e complexas em banco de dados transacional em uma máquina virtual porém em diferentes implementações, variando entre NFS (*Network File System*) com a própria máquina real, NFS com outra máquina real e *File System* da própria MV.

## 2 Experimento

O experimento foi executado em máquinas virtuais por meio do XEN e as imagens gerenciadas pelo OpenNebula [1][4]. Com isso, os dois ambientes foram executados com a mesma configuração da imagem, porém o HD do segundo cenário necessitou de mais disco devido ao TPC-H.

### 2.1 Cenário 1 – MPI [2] (MPICH2)

Na aplicação distribuída, foram criadas 4 máquinas virtuais com 1GB de memória, 1 CPU e 5GB de HD. Porém cada MV foi implantada em 2 máquinas reais (AMD Phenom 9600 / 9650 4 núcleos de 2.3Ghz, 4GB de memória, HDD 70GB), rodando o Opensuse 11.1 64 bits, kernel 2.6.27.45-0.1-xen como sistema operacional, Xen 3.3 para a virtualização das máquinas virtuais, e OpenNebula 1.4.0 para o gerenciamento e criação das MVs.

A fim de avaliar o quanto teve de trabalho na MV, foi selecionado o algoritmo de multiplicação de matrizes de ordem 500x500 escrito em fortran, e NPB [3] (NAS) *benchmark* classe C IS (*Integer Sort*) e CG (*Conjugate Gradient*). O NPB foi compilado com 2, 4, 8, 16, 32 e 64 processos, sendo que a quantidade de processos será divida entre as 4 máquinas virtuais.

**Tabela 1. Tabela de tempo de execução das aplicações distribuídas**

| Número de processos | IS Média | IS Desvio | CG Média | CG Desvio |
|---|---|---|---|---|
| 2 | 277,45 | 6,42 | 530,72 | 2,34 |
| 4 | 239,99 | 11,74 | 560,59 | 2,63 |
| 8 | 244,15 | 6,99 | 711,64 | 9,18 |
| 16 | 272,34 | 7,65 | 1798,30 | 61,46 |
| 32 | 328,11 | 12,44 | 2025,26 | 91,46 |
| 64 | 419,33 | 15,69 | 5739,96 | 121,28 |

Analisando a tabela 1, obteve-se duas conclusões. A primeira que o *benchmark* CG teve um desempenho melhor com poucos processos, o motivo é que o CG com pouca utilização de rede tem um desempenho mais eficiente [8]. A segunda análise do

benchmark IS demonstrou um desempenho mais eficiente com 4 processos, pois o *workload* foi divido em 4 MVs e teve como apoio a utilização de rede usando o DOM0 [8], sem o TPC/IP para influenciar negativamente no desempenho dos processos MPI.

## 2.2 Cenário 2 - DB2 e TPC-H

Na aplicação transacional, foi criada apenas 1 máquina virtual com 1GB de memória, 1 CPU e 25GB de HD em uma das máquinas reais. Como a versão do DB2 Express-C não permite particionamento do banco de dados, não foi possível configurar o banco de dados para funcionar em ambiente distribuído. Entretanto, foi possível implementar em diferentes sistemas de arquivos (*File System*): no primeiro teste, foi utilizado o *File System* da própria MV (ext3), no segundo, foi configurado o *File System* usando o NFS (*Network File System*) sincronizando com a máquina real e outro NFS com uma segunda máquina real para avaliar o comportamento forçando a comunicação de rede.

Na avaliação do desempenho das consultas *table scans* e complexas, foi aplicado o TPC-H, que é um forma de mensurar o desempenho de um banco de dados transacional (*benchmark*) que possui um script para a criação dos dados a serem populados nas tabelas do TPC-H. Para esse cenário, foi criado 2GB de dados, tendo uma quantidade de 7 milhões de registros, divididos em 8 tabelas.

As consultas utilizadas foram *table scan*, onde a cláusula "*where*" não foi aplicado nenhum campo com qualquer tipo de índice, fazendo assim uma leitura de todos os registros da tabela buscando os registros que satisfazia a consulta. Por outro lado, na complexa foram utilizados *subqueries* para avaliar subconjunto de registros para filtrar a consulta principal. Sendo assim, foi possível avaliar o desempenho de duas consultas com características distintas com objetivos diferentes, *table scan* necessita de IO de disco, e a consulta complexa requer mais processamento de CPU.

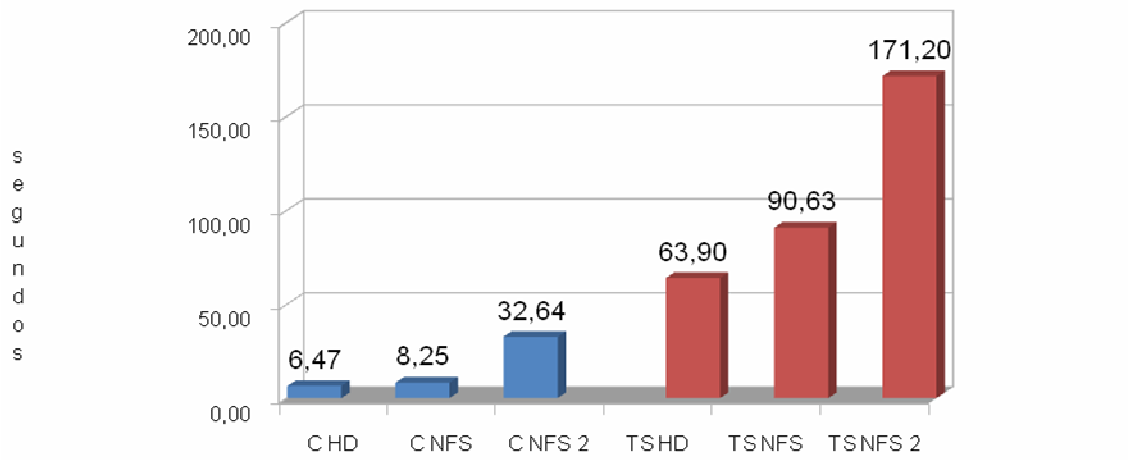



**Figura 1: Gráfico de desempenho das consultas *table scan* e complexas**

Segundo a figura 1, para as consultas "C HD" (Complexa HD) e "TS HD" (*Table Scan* HD), foi configurado um FS (ext3) na própria máquina virtual, para obter dados de execuções diretamente nesse diretório local. Já as *queries* "C NFS" (Complexa NFS) e "TS NFS" (*Table Scan* NFS), foi configurado um diretório NFS na máquina real, para avaliar o desempenho usando a rede com o DOM0 [8]. E por fim, as consultas "C NFS 2" (Complexa NFS em outra máquina real) e "TS NFS 2" (*Table Scan* NFS em

outra máquina real) foi configurado um diretório NFS em outra máquina real, forçando a comunicação via rede.

## 3 Conclusões

Esse trabalho teve como objetivo buscar trazer dados e análises sobre diferentes cenários com diferentes tipos de aplicações. No cenário com as aplicações distribuídas obteve-se duas respostas importantes, sendo que para o *benchmark* CG quanto menos processos forem utilizados, melhor vai ser o desempenho [7]. Contudo, no segundo cenário com o *benchmark* IS, o melhor cenário foi com a quantidade de processadores que não sofreram concorrência de processamento com outros processos. A partir de 4 processos, o MPI teve que dividir o processo entre as 4 máquinas virtuais.

Sobre o segundo cenário, pode-se concluir que se optar em utilizar NFS com a própria máquina real, é possível ter um desempenho bom, pelo fato de que a comunicação da rede ocorre através do DOM0 [8]. Entretanto, ao utilizar NFS com uma outra máquina real, o tempo de execução foi pior. Com o NFS é possível ter flexibilidade na migração do banco de dados com outras MVs ou até entre outros ambientes, podendo assim agilizar a migração de uma MV.

## 4 Referências


[1]Sotomayor , B. and Montero, R. S. and Llorente, I. M. and Foster, I. “Capacity Leasing in Cloud Systems using the OpenNebula Engine”

[2]Snir, M., Otto, S. W., Walker, D. W., Dongarra, J., and Huss-Lederman, S. “MPI: The Complete Reference.” MIT Press, Cambridge, MA, USA, 1995.

[3]NPB 3.3. Acessado em 21/05/2010. Disponível em: http://www.nas.nasa.gov/Resources/Software/npb.html

[4]OpenNebula. Acessado em 22/05/2010. Disponível em http://www.opennebula.org.

[5]MPICH2. Acessado em 27/05/2010. Disponível em http://www.mcs.anl.gov/research/projects/mpich2/about/index.php?s=about

[6]Bailey, D., Harris, T., Saphir, W., Van der Wijngaart, R, Woo, A., Yarrow, M., “NAS Parallel Benchmarks Version 2.4”, 12/1995.

[7]Lewis, J. G., Payne, D. G.. “Matrix-Vector Multiplication and Conjugate Gradient Algorithms on Distributed Memory Computers”

[8]Baruchi, A. Midorikawa, E T, “Influência do Algoritmo de Escalonamento Credit Scheduler no Desempenho de Rede”, WSO 2008.

# Sessão 3

# Alto Desempenho

# Análise de Escalabilidade e Determinação de Número Adequado de Processos para Aplicações Paralelas em Grids

**Samuel R. Silva[1], Hélio C. Guardia[1]**

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13.565-905 – São Carlos – SP – Brasil

samuel_rsilva@comp.ufscar.br, helio@dc.ufscar.br

***Abstract.*** *Finding the proper degree of parallelism for applications on computational grids is a non-trivial problem whose automatic obtainment is widely desired. This text presents an on-going work on the analysis of scalability factors of different applications classes aimed to build a heuristics, based on machine-learning techniques, to suggest the adequate number of processes for the execution of parallel applications on computational grids.*

***Resumo.*** *A determinação do grau de paralelismo adequado para aplicações em grid é um problema não trivial cuja obtenção automatizada é amplamente desejada. Este trabalho apresenta um estudo em desenvolvimento relacionado à escalabilidade de diferentes classes de aplicações, buscando-se estabelecer uma heurística, baseada em técnicas de aprendizado de máquina, para a determinação do número adequado de processos para execução de aplicações paralelas em grids computacionais.*

## 1. Introdução

A execução eficiente de aplicações paralelas em ambientes computacionais heterogêneos é uma tarefa não trivial. Algoritmos de escalonamento para tais ambientes são comumente aplicados sobre um modelo das aplicações, representado na forma de um Grafo Acíclico Direcionado (GAD), e seleciona as tarefas para execução de acordo com suas características de comunicação e processamento [1, 2].

Uma técnica de criação automática de GADs para aplicações MPI, desenvolvida neste grupo de pesquisa, auxilia na determinação de características das aplicações que interferem em seu comportamento [3,4]. Diferentes estratégias de predição de desempenho e de determinação do número apropriado de *threads* para uma aplicação também têm sido propostas [5,6,7], baseando-se em informações coletadas durante compilação e execução e na identificação de padrões entre as características de uma aplicação e uma base de conhecimento de aplicações previamente executadas.

Aparentemente, o mesmo tipo de estratégia pode ser utilizado para adequação do número de processos para ambientes computacionais distribuídos. Assim, o objetivo deste projeto é criar uma estratégia para estender abordagens de coleta de informação sobre aplicações em tempo de compilação e execução, a fim de determinar um método de análise de escalabilidade e predição de número adequado de processos para execução eficiente de aplicações paralelas em ambientes de grid.

## 2. Paralelização de Aplicações

A determinação do grau de paralelismo adequado para cada aplicação ainda é um

problema relevante. O número de processadores disponíveis pode ser usado por aplicações paralelas parametrizadas para explicitamente decidir como particionar suas atividades e por compiladores paralelizantes para determinar automaticamente o número de *threads* para a aplicação.

Considerando a multiplicidade de recursos heterogêneos e latências variáveis em grids, contudo, a mesma estratégia de paralelização não pode ser usada. Embora o uso do grid favoreça a maximização do paralelismo, a escolha do número adequado de nós depende da escalabilidade de cada aplicação. Complexas relações de dependência entre os dados se manifestam em muitas aplicações, fazendo com que o uso eficiente dos recursos paralelos dependa de fatores como estratégias de sobreposição de processamento e entrada/saída e da granularidade das tarefas. A utilização eficiente dos recursos disponíveis em um grid envolve determinar o grau de paralelismo adequado para cada aplicação, o que é um problema não trivial cuja obtenção automatizada é amplamente desejada.

O estudo do comportamento de aplicações, em especial de sua granularidade, pode indicar características que, associadas a estruturas que capturem as tarefas e comunicações das aplicações, como GADs, auxiliem na determinação do número adequado de nós a utilizar. Experiências anteriores deste grupo de pesquisa permitiram a criação automatizada de GADs correspondentes a aplicações paralelas, expressados seus módulos de processamento e suas comunicações, e permitem o escalonamento automático das aplicações em grids [2, 3, 4]. Para tanto, o conhecimento sobre as aplicações é extraído a partir do monitoramento de suas execuções com um número de nós definido pelo programador e não há extrapolações sobre variações desse parâmetro.

Assim, a proposta deste projeto é o estudo do comportamento e da relação computação/comunicação de diferentes classes de aplicações para se estabelecer uma heurística de sugestão do número adequado de processos/processadores para a execução de aplicações paralelas em grids. Para tanto, a heurística pretendida deverá usar técnicas de aprendizado de máquina (*Machine Learning*, ML), visando a associar características de aplicações com padrões já observados em outras aplicações.

Estudos similares foram realizados em ambientes monoprocessados e em ambientes multicore e ferramentas específicas foram propostas para tais ambientes [5-8]. Todavia, acredita-se que novos estudos sobre o modelo de tarefas e suas comunicações podem ser explorados também para se obter uma melhor atribuição de execuções em ambientes distribuídos de grades computacionais.

Um trabalho de destaque é o projeto Milepost GCC [7], que sugeriu o uso de aprendizado de máquina combinado com um compilador robusto e estável para, automaticamente, otimizar aplicações para melhor desempenho, tamanho de código, consumo de energia ou qualquer outra função objetivo desejada. Neste projeto, plugins não intrusivos estendem a funcionalidade do compilador GCC para controlar as heurísticas de otimização e para extrair as características de uma aplicação. Foram definidas 55 características e sua coleta ocorre em duas fases: primeiro, uma representação relacional do programa, a partir de sua representação interna, é extraída; em seguida, um vetor de características que codifica as várias estruturas de dados que representam o programa é calculado. O componente de ML, baseado na construção de um modelo probabilístico a partir de um conjunto de exemplos de treinamento, sugere, então, uma boa sequência de otimizações para o programa dado.

Modelos teóricos de caracterização de carga de trabalho já foram usados [8] para

mostrar que existe um número ótimo de nós para execução em sistemas de grid, que depende das distribuições dos tempos de computação e de comunicação. Observa-se que o número ótimo de nós se altera de acordo com a escolha de distribuição que modela as comunicações. Com um pequeno número de nós, o parâmetro dominante é o tempo de computação. O aumento da quantidade de nós faz com que os tempos de comunicação afetem o comportamento do sistema e o tempo de resposta.

## 3. Objetivos

O objetivo deste trabalho é a determinação automática de um número adequado de nós para a execução eficiente de aplicações paralelas em ambientes de computação distribuída. Os problemas relacionados são abordados usando mecanismos dedicados à predição do comportamento de aplicações, através de estudos de escalabilidade, da pesquisa de estratégias de associação de aplicações através de padrões de comportamento, bem como pelo uso de experiência prévia deste grupo de pesquisa na geração automática de GADs e no escalonamento automático de aplicações paralelas em ambientes de grid. Adicionalmente, o trabalho proposto direciona-se na busca dos seguintes sub-objetivos:

- Identificar um conjunto apropriado de características relevantes de aplicações, através da compreensão da influência de fatores como loops, saltos/desvios, ocupação da memória, estruturas de dados, primitivas de comunicação e otimizações de compilação, e também da influência de propriedades do sistema computacional alvo, tais como arquitetura de memória e sistema de comunicação;
- Adotar uma estratégia de extração e classificação de características das aplicações;
- Propor uma abordagem de combinação de características das aplicações com padrões de comportamentos já analisados.
- Avaliar o efeito dos parâmetros determinados na escalabilidade das aplicações

## 4. Resultados Preliminares

Estudos preliminares permitiram identificar uma série de características de aplicações paralelas como relevantes e influentes na escalabilidade, organizadas em três categorias e que podem ser extraídas da representação intermediária do compilador GCC usando-se a mesma abordagem adotada pelo projeto Milepost:

*a) geral*: contagem de instruções na representação intermediária (RI), de instruções load/store na RI e de saltos/desvios na RI;

*b) loops*: profundidade, contagem de iteração, quantidade de operações aritméticas, tipo de dependência de dado (fluxo, anti, saída), quantidade de declarações dentro do corpo do loop, quantidade de referências a arrays;

*c) comunicação*: modo (síncrono, assíncrono, bloqueante, não bloqueante), padrões de interação (um-pra-um, um-pra-muitos, muitos-pra-um, muitos-pra-muitos), acessos a recursos comuns e quantidade de bytes transmitidos, dada pelo tamanho dos buffers.

Além disso, a granularidade da aplicação, razão entre os tempos de computação e de comunicação, é fundamental na determinação de sua escalabilidade. Estes dados podem ser obtidos de um GAD resultante da execução da aplicação em um cluster.

O custo das comunicações, bem como as características de processamento, foram identificados como determinados pelos seguintes fatores do ambiente computacional:

*a)* composição do grid (supercomputadores, desktops, misto), quantidade de máquinas, quantidade e distribuição de processadores, quantidade de organizações virtuais (*Virtual Organizations*, VOs);

*b)* largura de banda dos enlaces de comunicações intra-VO e inter-VO;

*c)* largura de banda do barramento memória-CPU, frequência de clock do processador, quantidade de memória disponível em uma dada máquina ou conjunto de máquinas.

No estágio atual deste trabalho, pretende-se refinar a compreensão da influência das dependências de dados de uma aplicação em seu paralelismo e investigar a combinação das características identificadas com os GADs.

## Referências


[1] TOPCUOUGLU, Haluk; HARIRI, Salim; WU, Min-you. Performance-effective and low complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, Piscataway, USA, v. 13, n. 3, p. 260–274, 2002.

[2] RIOS, Ricardo et al. Análise de heurísticas para escalonamento online de aplicações em grade computacional. In VII WORKSHOP ON GRID COMPUTING AND APPLICATIONS, 2009, Recife, Brasil. **Anais do WCGA 2009.** Recife: SBC, 2009, p. 13-24.

[3] JACINTO, Daniele S.; RIOS, Ricardo A.; GUARDIA, Hélio C. Automatic modeling and grid scheduling MPI applications. In: WORKSHOP DE COMPUTAÇÃO EM GRADE E APLICAÇÕES, 2007, Belém. **Anais do Simpósio Brasileiro de Redes de Computadores e do Workshop de Computação em Grade e Aplicações**, 2007, v. 1, p. 75-86.

[4] JACINTO, Daniele S. **Escalonamento de aplicações paralelas**: de clusters para grids. 2007. 127 p. Dissertação (Mestrado em Ciência da Computação) – Centro de Ciências Exatas e da Terra, Universidade Federal de São Carlos, São Carlos, 2007.

[5] TOURNAVITIS Georgios et al. Towards a holistic approach to auto-parallelization: integrating profile-driven parallelism detection and machine-learning based mapping. In: CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, 2009, Dublin, Ireland. **Proceedings of the 2009 ACM SIGPLAN conference on Programming language, design and implementation.** New York, USA: ACM, 2009. p. 177-187.

[6] WANG, Zheng; O'BOYLE, Michael F.P. Mapping parallelism to multi-cores: a machine learning based approach. In: ACM SIGPLAN SYMPOSIUM ON PRINCIPLES & PRACTICE OF PARALLEL PROGRAMMING (PPOPP), 2009, Raleigh, USA. **Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming**. New York, USA: ACM, 2009, p. 75-84.

[7] FURSIN, Grigori et al. MILEPOST GCC: machine learning based research compiler. In: GCC DEVELOPERS' SUMMIT, 2008, Ottawa, Canada. **Proceedings of the GCC Developers' Summit.** Ottawa: 2009, p. 7-19.

[8] MUTTONI, Lorenzo et al. Optimal number of nodes for computation in grid environments. In: 12TH EUROMICRO CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING, 2004, A Coruña. **Proceedings of the PDP'04.** Los Alamitos: IEEE Comp. Society, 2004, p. 282-289.

# Atenuação do Gargalo no acesso a Disco em Sistemas Multiprocessadores

**Darlon Vasata[1], Liria Matsumoto Sato[1]**

**[1]Escola Politécnica da Universidade de São Paulo (EPUSP)**

{darlon.vasata,liria.sato}@poli.usp.br

***Abstract.*** *This work presents a study on disk access bottlenecks for multiprocessor architecture systems and also proposes a specification to solve this problem and its implementation. The developed strategy is called FPA. The developed strategy makes use of shared memory architecture and uses shared data buffers among several application threads, and also tries to optimize disk access time. Is allowed to just one application thread to read or write to the disk simultaneously, and in situations of writing, the processing thread operations run on parallel with disk operations.*

***Resumo.*** *Este trabalho apresenta um estudo sobre o gargalo no acesso a disco sob arquiteturas com sistemas multiprocessadores e processadores multicore e apresenta uma proposta de solução para o problema e sua implementação. A solução desenvolvida é chamada de FPA. A estratégia desenvolvida aproveita os recursos de memória compartilhada da arquitetura e utiliza buffers de dados compartilhados entre as diversas threads das aplicações, bem como tenta otimizar o acesso ao disco rígido. É permitido que apenas uma thread realize leituras ou gravações no disco por vez, e nas situações de escrita é feita a sobreposição entre o processamento e o envio dos dados ao disco.*

## 1. Introdução

Durante os últimos 20 anos, o aumento do desempenho nos discos rígidos tem sido significativamente menor do que o das Unidades Centrais de Processamento (CPU's). No período de Janeiro de 1996 a Janeiro de 2006 o aumento de desempenho das CPU's foi de 30 vezes, enquanto que os discos rígidos mantiveram uma escala de 1.3 vezes [Matthews et al. 2008].

A tecnologia atual de computadores disponível no mercado fornece processadores com diversos núcleos de processamento, ou mesmo máquinas com diversos processadores. Estas tecnologias permitem executar diversas tarefas simultaneamente, diferente do mecanismo que intercala as execuções das aplicações (*timesharing*), como acontece com os monoprocessadores. Com isto, tais tecnologias proporcionam um melhor desempenho em aplicações com suporte a paralelismo. Porém, como o acesso ao disco rígido é feito utilizando apenas um canal de comunicação, se diversas linhas de execução (*threads*) acessarem o disco simultaneamente, os dados obtidos não poderão ser fornecidos a todas as *threads* ao mesmo tempo. Desta forma, para aplicações que processam quantidades massivas de informações, acessam e armazenam seus dados a partir de arquivos, torna-se evidente a existência de um gargalo no acesso ao disco.

Outro fator agravante é o mecanismo físico de acesso às informações no disco, que acontece caso o acesso aos dados necessite de muitas movimentações no cabeçote de leitura/escrita do disco.

## 2. Objetivo

O presente trabalho objetiva uma solução para a problemática do acesso a disco em sistemas dotados de multiprocessadores, existente em aplicações *multithread*. Para tal, é desenvolvida uma estratégia e implementada em uma biblioteca denominada de *File library for Parallel Access* (FPA), de forma que toda a estratégia possa ser utilizada de forma portável e transparente pelos programadores de aplicações *multithread*.

A idéia geral do trabalho é proporcionar uma otimização no acesso a disco realizada em nível de aplicação, devido a que implementações neste nível são menos complicadas do que implementações em nível de *kernel*, implicando em custo geral menor de desenvolvimento, maior portabilidade e facilidade de implantação [Silberschatz 2004].

## 3. Estratégia proposta e implementação

A estratégia do trabalho busca a solução para o problema do acesso ao disco sem alterações na arquitetura ou *hardware* adicionais, dispondo apenas de técnicas em *software*, possibilitando que esta possa ser utilizada em quaisquer aplicações construídas utilizando a linguagem de programação C e suporte a Pthreads.

A implementação da FPA consiste em utilizar estratégias de otimização no acesso a disco, como permitir o acesso de apenas uma *thread* por vez e a utilização de buffers internos à biblioteca para os dados das aplicações.

O armazenamento temporário de dados possibilita a sobreposição entre o processamento da aplicação e a gravação de dados ao disco, onde a aplicação não precisa aguardar pelo envio dos dados e assim continuar o seu processamento. Posteriormente, quando toda a região temporária estiver cheia, os dados poderão ser enviados ao sistema de armazenamento, nas situações de escrita. Para os casos onde existe leitura de dados, toda a região temporária pode ser preenchida de uma única vez. A prática de utilizar operações de entrada e saída com quantidades de dados maiores em menores quantidades ao invés de diversas operações com quantidades pequenas de dados implica em uma sobrecarga menor ao sistema operacional, enquanto também faz um melhor aproveitamento das operações ao disco. Alem disso, outra vantagem é que uma outra *thread* pode assumir o gerenciamento da leitura/escrita dos dados e definir o tempo em que estes devem ser atualizados com o disco, sem que a aplicação principal interfira neste gerenciamento. Isto permite com que seja realizado um melhor aproveitamento dos movimentos do cabeçote de leitura e escrita do disco.

## 4. Resultados obtidos

Os testes da biblioteca FPA foram executados em uma máquina do *cluster Open Modeller* (OM), na EPUSP. As máquinas do *cluster* são compostas por processadores Dual Quadcore Intel® Xeon®, com *clock* de CPU de 2.0 Gigahertz, arquitetura de 64 bits, memória RAM de 8 Gigabytes e disco rígido com padrão de conexão Serial ATA. O sistema de arquivos utilizado foi o XFS, junto com o sistema operacional SUSE Linux Enterprise Server 10.

Na execução dos testes foram levados em considerações três fatores principais: o tamanho do arquivo, a quantidade de processamento existente na aplicação e o número de *threads* utilizadas. Para que a quantidade de processamento existente nas aplicações pudesse ser mensurada, nos códigos de teste é inserida uma rotina que envolve operações de multiplicação e divisão, e a quantidade de vezes em que estas operações são executadas é utilizada como referência nos resultados. Os tempos de execução obtidos pela FPA são comparados com os tempos obtidos com a execução utilizando os *streams* com as operações `fread` e `fwrite`, disponíveis na LIBC.

O formato dos testes realizados possuem o intuito de verificar o desempenho da aplicação sob uma situação em que diversas *threads* acessam o mesmo arquivo, porém cada uma delas acessa uma região diferente, como pode ser observado na Figura 1.

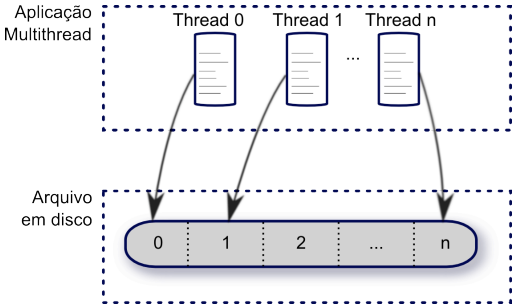



**Figura 1. Formato dos testes realizados**

Nos testes envolvendo a leitura e escrita de dados ao disco ocorre uma diminuição da concorrência ao disco com a utilização da FPA, como pode ser visto no gráfico apresentado na Figura 2.

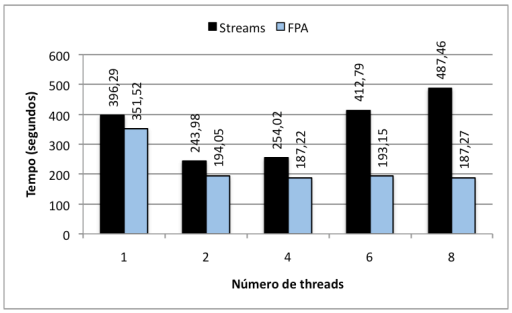



**Figura 2. Leitura e escrita com arquivos de 4 GB e processamento de 5000 iterações, utilizando número de *threads* diferentes**

A divisão do tempo total de processamento da aplicação é aproveitada pelos *streams* até os testes utilizando 2 *threads*. A partir desse ponto a concorrência ao disco aumenta a ponto de inviabilizar a aplicação paralela, pois com 6 e 8 *threads* o tempo de execução é maior do que o tempo gasto na aplicação sequencial. Ainda na Figura 2 é possível visualizar que na maioria dos casos testados a FPA possui ganho de desempenho, em relação aos *streams*. Tal ganho deve-se principalmente à utilização dos buffers, pois os dados são gerenciados em memória e mais tarde o buffer completo será enviado ao disco. Isto implica em um menor numero de chamadas de sistema requisitando operações de E/S e assim resultando em uma sobrecarga menor ao sistema operacional.

A sobreposição entre o processamento da aplicação é melhor aproveitada pela FPA, como pode ser observado na Figura 3. Nesta situação observada, existem as otimizações

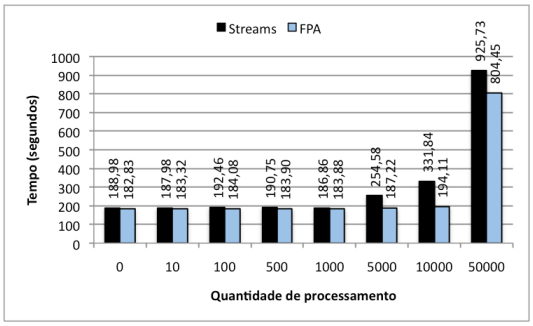



**Figura 3. Leitura e escrita com arquivo de 4 GB e 4 *threads*, utilizando diferentes quantidades de processamento na aplicação**

em ambas operações de leitura e escrita, e nas situações onde as operações de leitura não apresentam ganho de desempenho, as operações de escrita são vantajosas, e vice-versa. Com isto, o resultado final de ambas operações torna-se vantajoso com o uso da FPA, inclusive para o caso sequencial.

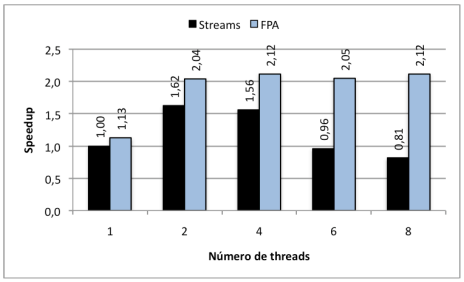



**Figura 4. *Speedup* da leitura e escrita, utilizando arquivos de 4 GB e processamento de 5000 iterações**

Na Figura 4, observa-se o ganho de desempenho da FPA em relação aos *streams*. Como a leitura do disco é feita preenchendo todo o buffer em memória, existe a vantagem da menor sobrecarga ao sistema operacional, que também acontece na escrita dos dados.

## 5. Conclusões

A estratégia definida para otimização no acesso a disco apresentou ser uma alternativa vantajosa, como foi visto nos resultados. Estes indicaram ganho de desempenho da FPA em relação ao uso dos *streams* da LIBC. O número de *threads* presente na aplicação influencia no comportamento da FPA, afinal com um número maior destas a concorrência ao disco tende a aumentar. Porém, como a FPA utiliza uma *thread* adicional para o envio dos dados ao disco, caso todas as unidades de processamento estejam ocupadas, esta *thread* pode ocasionar em concorrência pela CPU com as *threads* da aplicação, podendo levar a uma perda de desempenho nessas situações.

## Referências bibliográficas

Matthews, J. et al. (2008) “Intel® turbo memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems”, ACM Transactions on Storage, New York, ACM, v. 4, n. 2, p. 4.

Silberschatz, A., Galvin, P.B., Gagne, G. (2004), Fundamentos de sistemas operacionais, LTC – Livros técnicos e científicos editora S.A., 6ta edição.

# Finding Fractional Identities in Multiple Sequences Using a Fast Parallel Algorithm

**Evandro A. Marucci[1], Geraldo F. D. Zafalon[1,2], Aleardo Manacero[1], Liria M. Sato[2], José M. Machado[1]**

[1]Departamento de Ciências de Computação e Estatística
Universidade Estadual Paulista - UNESP

[2]Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica - Universidade de São Paulo - USP

{marucci,zafalon,liriasato}@gmail.com, {aleardo,jmarcio}@ibilce.unesp.br

***Abstract.*** *The calculation of similarities is an operation widely used in Bioinformatics. Usually made between pairs of sequences, it determines the relative amount of common residues between them. The fractional identity is a measure of similarity used in many sequence alignment tools. Due to the growing of volume of comparative data, the interest in parallel computing is increasing remarkably. The implementation of a parallel algorithm for finding fractional identities in multiple sequences is proposed in this paper. Tests have shown that the algorithm presents a very good scalability and a nearly linear speedup.*

## 1. Introduction

The use of sequence comparison methods has been growing remarkably in recent years, in response to the volume increasing of available genomic data. Consequently, the development of new approaches that reduce the processing time are fundamental to the improvement of this area.

In general, the sequences comparison starts with the search of similarity degree between pairs of sequences. This search is made by methods which may require or not that the sequences are aligned. A method widely used by the most of sequence alignment tools and which requires a prior alignment is the fractional identity [Edgar 2004a] method. CLUSTALW [Thompson et al. 1994] and MUSCLE[Edgar 2004b] use this method in one of their steps.

We propose in this paper a parallel algorithm for fractional identity method and show implementation information [1]. This algorithm can be used in the development of any parallel tool which needs a stage of fast refinement with a great biological accuracy.

## 2. Fractional Identity

Given an alignment, we ignore all positions with gaps and, in the other positions, we calculate the amount of them with equal residues in relation to the total amount of positions

[1]This work was partially supported by the São Paulo Research Foundation (FAPESP - Brazil) under Grant No. 06/59592-0
Corresponding author: Geraldo F.D. Zafalon

with any kind of residues. Figure 1 gives an example of the fractional identity calculation. In this figure, the aligned pair of sequences has 9 positions. Among them, only positions 1, 2, 4, 5, 6, 7, 9 and 10 have no gaps. Positions 2, 4, 6, 7 and 9, in turn, are those having equal residues in both sequences. Thus, 8 positions without gaps and 5 positions with equal residues. The fractional identity between these sequences is therefore $5/8$.

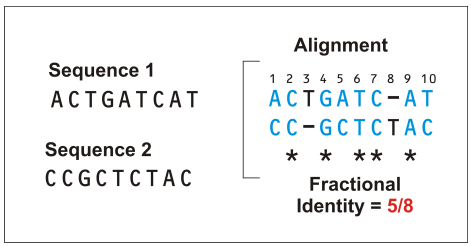



**Figure 1.** *Fractional identity calculation between two sequences.*

## 3. Parallel Algorithm

This algorithm dynamically divides the computation among the processors through a master-slave approach. This parallelism is performed by distributing the similarity calculation of sequence pairs among the available slaves. This is possible because each fixed pair similarity calculation is independent from the remaining pair similarity calculations. The flowchart of this algorithm is shown in figure 2.

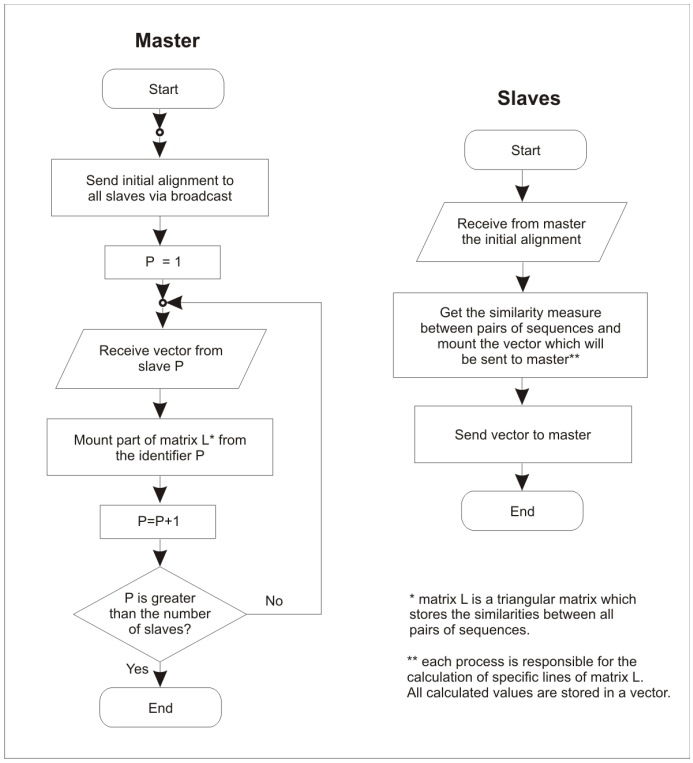



**Figure 2.** *Parallel algorithm flowchart for the calculation of fractional identities in multiple sequences.*

Initially, the master sends all sequences by broadcast to all slaves. The task distribution is based on the processor identifier, assigning to each slave the calculation of

specific lines in the similarity triangular matrix. This matrix is obtained as exemplified in figure 3. Each slave initially calculates the line corresponding to the identifier, in a $p$-step loop, where $p$ is the number of processors. In the example, we have eight sequences, and, hence, a seven by seven matrix. The first slave is responsible for the calculation of the lines from sequences 1, 4 and 7. The second one by lines of sequences 2 and 5 and the third by lines of sequences 3 and 6. The gray values are the results obtained in other slaves; they will be joined in the master for the creation of the similarity matrix. Although simple, this scheduling tends to a good load balancing for any number of sequences involved.

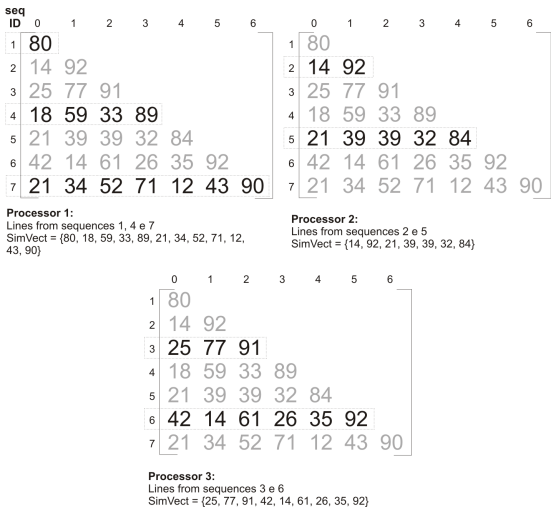



**Figure 3.** *Example of how the calculation of the similarity matrix is distributed between the slaves.*

## 4. Tests and Results

In order to verify the proposed algorithm performance, we executed tests with four different datasets. For each dataset, we verified the algorithm scalability when executed in a growing number of machines. All tests were performed on a Beowulf cluster.

Some executed tests are showed in the figure 4. There are four graphics listed from 1 to 4. Basically, in this figure are showed some graphics which have in the horizontal axis the number of processors and in the vertical axis the running time in seconds. On the top of each graphic are placed some parameters of the sequence datasets used in the tests. The parameter *num* represents the number of sequences used in the execution, the parameter *max* represents the maximum number of residues found among the sequences of dataset and the parameter *avg* shows the average number of residues found among the sequences of dataset.

The first three graphics illustrated in figure 4 show an almost linear speedup for tests with more than 500 sequences. The first one with 567 sequences, the second with 988 sequences and the third one with 2000 sequences.

The last graphic (number 4) shows a constant performance, regardless of the number of machines, for an entry with few sequences. In this case, the sequential implementation of the algorithm is so fast for that input (approximately 1 second) that the speedup achieved with tasks division is close to the time spent with message exchanges.

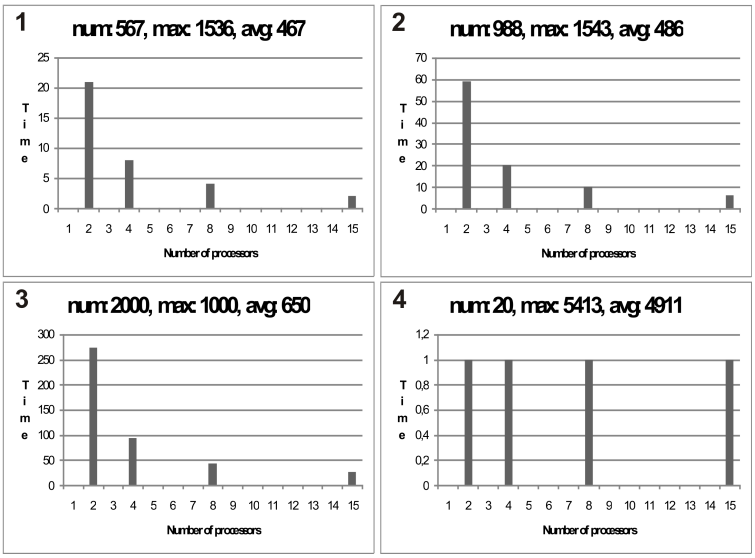



**Figure 4.** *Running time of the parallel algorithm for 2, 4, 8 and 15 processors with four different datasets.*

The performance achieved with the addition of 14 processors is more than 10 times higher. In tests we can realize a nearly linear speedup independent from the dataset used and the number of processors in the cluster.

## 5. Conclusions

In this article, we propose and measure a parallel strategy to calculate similarities between multiple sequence pairs by using fractional identities. This calculation is used, for instance, in multiple sequence alignment tools. The proposed parallel algorithm has been implemented for distributed memory systems, due to the wide use of Beowulf clusters in genomic research.

The performed tests show that the algorithm presents a good scalability and a nearly linear speedup. With the use of 14 processing nodes (slaves), the system achieved a 10 times more speedup. This speedup is justified by the total independence of the scheduled tasks and by the good load balancing obtained with the merge distribution of triangular matrix lines. Additionally, the communication cost is minimized because the processed data in slaves are sent to the master in a single message.

## References

Edgar, R. C. (2004a). Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Research*, 32(1):380–385.

Edgar, R. C. (2004b). Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1).

Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680.

# Alocação de recursos distribuídos em Grade de computadores

**Francisco Ribacionka**[1] **Liria Matsumoto Sato**[1] **Luciana Arantes**[2]

[1] Escola Politécnica da Universidade de São Paulo

[2]Laboratório de Informática de Paris 6 (LIP6), Universidade Pierre et Marie Currie (Paris 6)

fribacio@usp.br liriasato@gmail.com Luciana.arantes@lip6.fr

***Abstract.*** *This article describes an algorithm for distributed resource allocation, which is an extension of the algorithm presented by Raynal [Raynal, 1991]. This algorithm can be applied in clusters or in a Grid environment for parallel applications. It is a decentralized model in the decision of who will receive the resources. When a request can not be fully met, a queue is formed and organized in a logical clock, with resource reservation.*

***Resumo.*** *Este artigo descreve um algoritmo de alocação de recursos distribuídos, que é uma extensão do algoritmo apresentado por Raynal [Raynal, 1991]. Este algoritmo pode ser aplicado em clusters ou em um ambiente de Grade para aplicações paralelas. É um modelo descentralizado na tomada de decisão de quem vai receber os recursos. Quando uma solicitação não pode ser atendida integralmente, uma fila é formada e organizada por um relógio lógico, com reserva de recursos.*

## 1. Introdução

Este algoritmo tem como objetivo alocar recursos distribuídos para uso simultâneo ou para processamento paralelo. Estes recursos podem estar em clusters separados geograficamente e participantes de uma organização virtual em Grade Computacional [Foster, 2001], [Berman, 2003]. É uma estratégia de escalonamento que pode ser aplicada em uma Grade Computacional ou em uma Nuvem (Cloud Computing).

Computação em grade tem como objetivo criar um computador virtual composto por uma coleção de sistemas heterogêneos ligados por rede, compartilhando recursos de armazenamento e de processamento [Ferreira, 2003, Arafah 2006]. É vista como uma infraestrutura que provê recursos para a criação de organizações virtuais (VO) que consiste de recursos geograficamente distribuídos mas que aparenta ter a função de uma única organização com um objetivo único.

Raynal [Raynal, 1991] mostrou a utilização de M recursos compartilhados por N processos, onde, em um mesmo instante, cada um destes recursos pode estar executando um processo. Diferentemente da proposta de Raynal, o algoritmo descrito neste trabalho discute a utilização de M recursos, distribuídos por K clusters, e que podem ser utilizados por N processos. A alocação de recursos é feita por reserva, ou seja, se um pedido não pode ser atendido imediatamente, entra em uma fila ordenada por um

relógio lógico [Lamport, 1978]. Este mecanismo promove o atendimento das solicitações, considerando a ordem de entrada na fila e evitando a ocorrência de *deadlocks* e *starvation*.

**2. Modelos de escalonamento de jobs**

Nos artigos estudados verificou-se que basicamente há dois modelos para escalonamento de Jobs em grades computacionais, ou Grades: centralizado e distribuído. No modelo centralizado, todas as informações ficam em um site, que é responsável pela distribuição de jobs para todos os sites que a ele estejam ligados. No modelo distribuído, cada site que compõe o grade possui o seu sistema escalonador de Jobs, e a forma da tomada de decisão de quem vai executar o job varia de acordo com o algoritmo desenvolvido [Subramani, 2002], [Schopf, 2004].

Leal e demais autores [Leal, 2009] apresentam um modelo descentralizado para o esquedulamento de tarefas independentes em Federações de grades de computadores. Este modelo consiste em um conjunto de meta-escalonadores em cada um dos grades pertencentes à Federação. Um meta-escalonador no topo da hierarquia desta federação possui informações genéricas sobre a Federação, e quatro algoritmos são apresentados com o objetivo de maximizar a utilização dos recursos da Federação: Objetivo estático, objetivo dinâmico, objetivo estático com esquedulamento avançado e objetivo dinâmico com esquedulamento avançado. Estes quatro algoritmos são baseados em um modelo de performance da estrutura que forma a Federação de Grades, não o seu estado.

Nilton [Nilton, 2009] propõe e implementa o sistema centralizado GCSE (Grade Cooperative Scheduling Environment) que provê uma estratégia de escalonamento cooperativo para usar eficientemente os recursos distribuídos por vários clusters e computadores, todos conectados a redes de comunicação pública.

Chao-Chin [Chao-Chin, 2010] propõe um algoritmo genético para submissão de Jobs que pode ser aplicado em um ambiente de grade que tenha tolerância a falhas para entrada de Jobs, migração de Jobs com ou sem checkpoint e mecanismos de replicação de Jobs.

**3. Alocação de recursos distribuídos**

Nesta seção descreve-se o algoritmo proposto. É uma extensão do algoritmo apresentado por Raynal [Raynal, 1991] para Grade. É um modelo descentralizado na tomada de decisão de quem vai obter recursos para processamento. Todos os clusters participantes tem um mapa de disponibilidade de recursos disponíveis. Os possíveis conflitos de solicitações simultâneas são resolvidos pelo uso do relógio lógico, e a fila de solicitações é feita por reserva de recursos.

O algoritmo tem três rotinas básicas: Pedindo_Recursos, Recursos_Solicitados e Liberando_Recursos. A rotina Pedindo_Recursos é acionada por quem precisa de recursos. Nela há a atualização do Mapa de Disponibilidade para a verificação se algum

cluster participante do Grade pode atender integralmente sua solicitação. Se não, verifica se, somando recursos de mais de um cluster, ele pode ser atendido. Se não, envia um pedido a todos para a solicitação ficar na fila de cada um. Esta rotina está ilustrada na figura 1.

```
Pedindo_Recursos
{       Atualiza o Mapa de Disponibilidade
        Verifica se pode ser atendido integralmente por um único participante
        Se não, verifica se somando recursos de mais de um cluster pode ser atendido
        Se não, coloca seu pedido na própria fila e envia um pedido a todos para ficar na
        fila de cada um
}
```

Figura 1 - Rotina Pedindo_Recursos

A rotina Recursos_Solicitados, representada na figura 2, é enviada para quem pode atender à solicitação feita na rotina Pedindo_Recursos, e decide quem pode utilizar os recursos disponíveis. Quando acionada, verifica se há uma reserva prioritária ao pedido que acaba de chegar. Caso não exista uma reserva prioritária, é feita a liberação de recursos para quem solicitou após o recebimento de confirmação de autorização de todos os participantes. Caso contrário, o pedido é colocado na fila local de cada cluster que recebeu esta solicitação.

```
Recursos_Solicitados
{       Se não existe uma reserva prioritária então pede confirmação de todos
        Senão coloca o pedido na fila local
}
```

Figura 2 - Rotina Recursos_Solicitados

A rotina Liberando_Recursos é usada quando um cluster disponibiliza recursos de terceiros (e dele mesmo) que ele estava utilizando. Se a fila local não está vazia, uma confirmação de todos é solicitada para ser feita a liberação ao primeiro da fila (a fila é ordenada pelo relógio lógico). Esta rotina está ilustrada na figura 3.

```
Liberando_Recursos
{       Se a fila local não está vazia então pega o 1o. da fila e solicita confirmação de
todos para liberar recursos}
```

Figura 3 - Rotina Liberando_Recursos

O Mapa de Disponibilidade (MD) é um vetor com o número de recursos disponíveis em cada cluster. Ele é atualizado na rotina Atualiza_MD, que é chamada sempre que a rotina Pedindo_Recursos é acionada. Na rotina Atualiza_MD também é feito o sincronismo dos relógios lógicos. A figura 4 mostra esta rotina.

```
Atualiza_MD
{       Pede a todos quantos recursos livres tem
        Atualiza o relógio lógico       }
```

Figura 4 - Rotina Atualiza_MD

3

## 4. Conclusão

O algoritmo descrito neste trabalho é uma proposta de extensão do algoritmo apresentado por Raynal [Raynal, 1991]. É uma proposta de solução descentralizada para alocação de recursos distribuídos, e por ser descentralizada, tem-se como uma das vantagens a facilitação de implementação de mecanismos de tolerância a falhas, que deverá ser realizada em um trabalho futuro. Outra vantagem é a escalabilidade, permitindo a inclusão de novos clusters na Grade como também a exclusão de outros.

## 5. Bibliografia


**[Arafah, 2006]** *Grid computing: a STOPE view*, International Journal of Network Management, Arafah M. A, (2006).

**[Berman, 2003**], *The grid: past, present, future, in Grid Computing: Making the Global Infrastructure a Reality*, Berman et al, (2003), Wiley.

**[Chao-Chin, 2010]** *An integrated security-aware job scheduling strategy for large-scale computational grids*, Elsevier: Future Generation Computer Systems 26, Chao-Chin Wu (2010).

**[Ferreira, 2003]** *Introduction to Grid Computing with Globus*, IBM Redbooks, Ferreira, L, et al, (2003).

**[Foster, 2001]** *The Anatomy of the Grid*, Ian Foster, Carl Kesselman, Steven Tuecke, Proceedings of the 7th Internationa Euro-Par converence Manchester on Parallel Processing, pages 1-4, 2001

**[Lamport, 1978]** *Time, Clocks and the Ordering of Events in a Distributed System*, Leslie Lamport, Communications of the ACM, July 1978, Volume 21, Number 7

**[Leal, 2009]** *A decentralized model for scheduling idependent task in Federated Grids*, Elsevier: Future Generation Computer Systems 25 (2009) 840-852

**[Nilton, 2009]** *Um ambiente de monitoramento de recursos e escalonamento cooperativo de aplicações paralelas em grades computacionais*, tese de doutorado, Escola Politécnica, USP

**[Raynal, 1991]** *A distributed solution to the k-out of-M resources allocation problem*, Michel Raynal, Institut de Recherche en Informatique Et Systemes Aleatoires, 1991

**[Schopf, 2004]** *Ten Actions When Grid Scheduling*, Jennifer M. Schopf, Grid Resource Management: State of the Art and Future Trends, p. 15-23, 2004

**[Subramani, 2002]** *Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests*, Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan and P. Sadayappan, Proceedings of 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), July 2002

# Sessão 4

# Computação em Grade

# Acesso e Compartilhamento de Bancos de Dados Heterogêneos Integrados Através de Grade Computacional

**Fernando Ryoji Kakugawa, Liria Matsumoto Sato, Mathias Santos de Brito**

Departamento de Engenharia da Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, tv 3, n 158 – São Paulo – Brasil

{fernando.kakugawa,liria.sato,mathias.brito}@poli.usp.br

***Resumo.*** *A necessidade em integrar bancos de dados entre diferentes organizações é um tema de pesquisa atual com diversas questões a serem resolvidas visto que a heterogeneidade envolvida entre os diversos bancos dificulta esta tarefa. Este trabalho propõe uma forma de integração e compartilhamento de dados entre diversas organizações utilizando uma infraestrutura baseada em computação em grade. Este sistema possibilita o acesso a diversos bancos de dados, mantendo a complexidade envolvida para o acesso e ao esquema lógico transparente ao usuário final.*

## 1. Introdução

Bancos de dados normalmente são concebidos para atender a um domínio específico de uma aplicação. Normalmente, cada instituição possui seu sistema de informação projetados de forma autônoma, procurando atender necessidades específicas, sem se preocupar com hardware, software, padrões ou formas de integração tornando o ambiente heterogêneo. Integrar dados armazenados em diferentes locais é um desafio que envolve diversos riscos de inviabilidade devido à complexidade envolvida [Barbosa, Porto e Melo, 2002].

Neste contexto, o conceito de computação em grade [Foster e Kesselman, 2004] vem sendo muito difundido nos últimos anos, tanto no meio acadêmico como na indústria. O maior objetivo deste conceito é atender a necessidade de utilizar sistemas de computação distribuída, possibilitando o compartilhamento e a coordenação de diversos recursos heterogêneos, distribuídos em diferentes Organizações Virtuais [Foster, Kesselman e Tuecke, 2001].

Este trabalho apresenta parte de uma infraestrutura de integração e compartilhamento [Brito, Kakugawa, Sato, *et al.*, 2009] para diversos bancos de dados heterogêneos dispersos geograficamente, utilizando a tecnologia de grades computacionais, respeitando cada política administrativa local.

## 2. Descrição do Sistema para Integração de Dados

Este trabalho propõe um sistema para integração entre bancos de dados relacionais heterogêneos de esquemas lógicos distintos realizando uma integração horizontal de dados. O sistema está baseado na tecnologia de computação em grade, onde a partir de bancos oferecidos como recursos computacionais, é oferecida uma interface para o acesso, deixando transparente a complexidade relativa à localização dos diversos bancos distribuídos e a diferença entre os seus esquemas lógicos. Tais características permitem fácil escalabilidade ao sistema, tanto na inserção como na remoção de bancos, pois

requisições enviadas a um Servidor de Acesso que não disponibiliza o banco de dados desejado pode ser encaminhada para o Servidor de Acesso que disponibiliza o banco e delega esta requisição. Esta característica faz com que a aplicação final não necessite de alterações caso haja modificações na base de dados. A Figura 1 ilustra a arquitetura do sistema assim como os seus componentes.

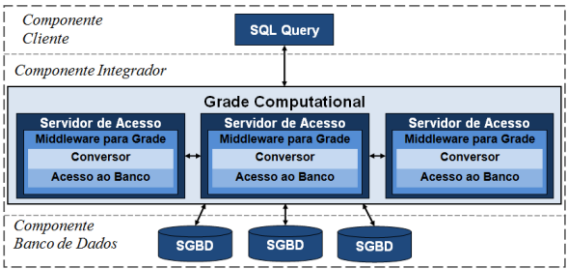



**Figura 1 – Arquitetura para o sistema proposto**

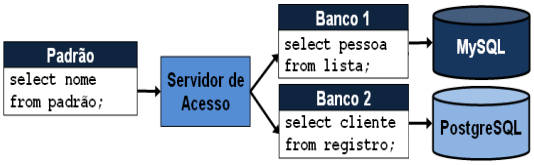



**Figura 2 - Execução de consulta**

Para o tratamento da heterogeneidade entre os bancos, o Servidor de Acesso utiliza um conversor de consultas que a modifica conforme características do banco desejado, convertendo uma consulta SQL a partir de um formato padrão para um formato compreensível ao banco de dados de destino. Para ilustrar este tipo de acesso, a Figura 2 mostra uma consulta SQL enviada a dois bancos de dados distintos. O aspecto atraente desta proposta está na localização das configurações de acesso e do conversor de consulta, que estão situadas no Servidor de Acesso onde o banco de dados está disponibilizado, desta forma requisições de outros bancos podem ser delegadas aos Servidores de Acesso onde os bancos estão disponibilizados. As subseções a seguir irão descrever cada um dos componentes da arquitetura, ilustrado na Figura 1.

## 2.1 Componente Banco de Dados

É responsável pelo acesso, persistência, manipulação e recuperação dos dados nas máquinas. Em essência, são os sistemas de bancos de dados, autônomos e dispersos geograficamente que serão integrados ao sistema desenvolvido, fornecendo suporte aos principais SGBDs relacionais do mercado, como MySQL, DB2, Oracle, Postgresql entre outros.

## 2.2 Componente Cliente

É responsável pelo acesso do cliente ao sistema de integração, onde é estabelecido um conjunto de tarefas (*activity*) no formato de um fluxo de trabalho (*workflow*) e o submetem para a execução nos Servidores de Acesso. As *activities* podem realizar diversos tipos de tarefas como consultas, descrever um recurso que deseja utilizar, solicitar o formato de retorno de uma consulta enviada, entre outros. O componente é compatível com o *driver* JDBC OGSA-DAI [Brito e Sato, 2008].

## 2.3 Componente Integrador

O componente integrador está baseado no *middleware* Globus Toolkit versão 4 (GT4) [Foster, 2005] e o *middleware* OGSA-DAI [Hong, Hume, *et al.*, 2007]. Este componente é responsável por receber um *workflow* do cliente e coordenar a sua execução. Para que este componente funcionasse conforme a proposta deste trabalho, foram desenvolvidas novas *activities* para o OGSA-DAI, que possibilitassem o acesso a

bancos de dados heterogêneos. As subseções a seguir descrevem os mecanismos desenvolvidos para o acesso aos bancos e como é realizada a conversão de consultas.

### *2.3.1 QCON-G (Query CONverter for Grid)*

O QCON-G é um conjunto de *activities* desenvolvido para prover o acesso simultâneo a bancos de dados relacionais com esquema lógico diferente, em um ambiente de grade computacional. Após o envio de uma consulta SQL ao recurso desejado, verifica-se quais os bancos envolvidos, onde cada banco está localizado, qual o seu esquema de dados local para que a consulta seja convertida e em seguida, encaminhada de forma adequada ao SGBD onde o banco de dados está residindo. Para a conversão de consulta, as *activities* utilizam o QCON, que será descrito a seguir.

### *2.3.2 QCON (Query CONverter)*

Este componente possui uma arquitetura baseada em cinco módulos, composta por Analisador, responsável pela análise léxica e sintática da consulta, Decomposição, que fragmenta a consulta em *tokens*, Mapeamento, onde através do Repositório de Esquema de Dados verifica quais os *tokens* correspondentes para o banco de dados de destino, assim o Conversor de posse dessas informações realiza a conversão na consulta. O esquema de cada banco de dados deve ser informado em um arquivo de configuração que está localizado no Repositório de Esquema de Dados.

## 3. Análises e Resultados

Para os testes foram utilizadas duas máquinas como Servidores de Acesso, interligadas através de uma rede *ethernet*, e uma terceira máquina com a aplicação cliente através de um roteador *wireless*. Cada Servidor de Acesso possui dois bancos de dados, com nomes de bancos, tabelas e colunas distintas entre si. Para todas elas foram enviadas a mesma consulta SQL em um formato que denominamos de esquema padrão. Os testes T1, T2, T3 e T4 utilizou um banco de dados, T5 e T6 utilizou dois, T7 e T8 utilizou três e T9 utilizou quatro bancos de dados para realizar uma consulta simultânea. O método definido para os testes foi comparar o tempo de execução de uma consulta (SELECT) utilizando o OGSA-DAI sem conversão de consultas com os tempos do QCON-G. É importante comentar que o OGSA-DAI não oferece suporte a bancos de dados que possuam esquema lógico diferente entre eles. Apesar de possuir um componente denominado DQP [Alpdemir, Mukherjee, *et al.*, 2003], este realiza um tipo de integração diferente ao proposto por este trabalho.

Observando os gráficos a seguir, foi verificada que para o acesso a um único banco de dados a diferença entre os tempos de acesso não foi maior que 3,85%, mostrando que o *overhead* causado pela conversão de consultas não inviabiliza o uso do QCON-G. A partir do teste T5, cujos resultados são apresentados na Figura 3, é verificada uma tendência de aumento no tempo de execução da consulta para o OGSA-DAI, enquanto que para o QCON-G o tempo de execução fica constante em torno de 800 ms, conforme mostra a Figura 4.

A degeneração no tempo verificada na Figura 3 a partir do teste T5 é atribuída ao fato do OGSA-DAI não possuir suporte para bancos de dados com esquema lógico diferente. O programador da aplicação cliente fica encarregado de todo o controle para utilizar o banco desejado em um Servidor de Acesso, resultando em aproximadamente 40% a mais de linhas de códigos para cada banco de dados adicionado para consulta.

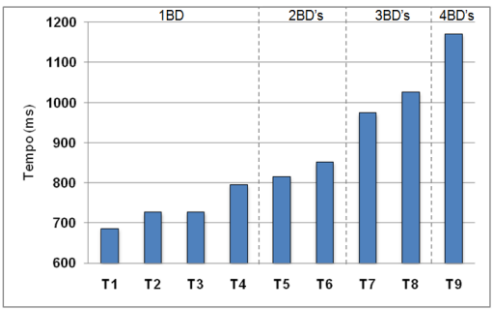



**Figura 3 – Tempo de execução utilizando OGSA-DAI**

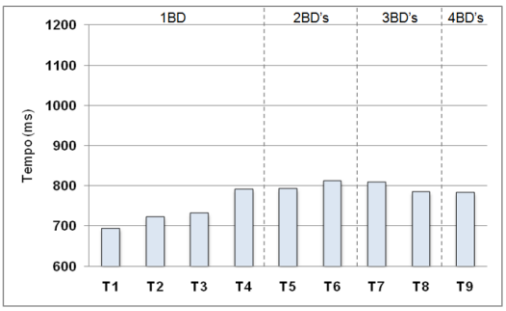



**Figura 4– Tempo de execução utilizando QCON-G**

## 4. Conclusões

O QCON-G mostrou que é um caminho viável para a integração de bancos de dados heterogêneos com esquema lógico diferente, visto que o OGSA-DAI não oferece esse recurso. Nos testes realizados, verificou que o *overhead* causado pela conversão de consulta não é tão significativo a ponto de inviabilizar o uso do QCON-G.

O benefício da conversão está em oferecer suporte para o acesso aos diversos bancos de dados distintos, deixando a aplicação cliente isento de alterações caso haja a adição ou a remoção de bancos. Isto é possível devido à opção adotada para que as configurações para o acesso simultâneo aos bancos de dados e o tratamento utilizando a conversão de consulta fiquem localizados no Servidor de Acesso que disponibiliza o banco de dados na grade computacional. Esta opção permite transparência neste acesso, a respeito da localização e do esquema lógico de cada um dos bancos presentes no sistema, oferecendo ao programador uma visão integrada e uniforme para o acesso utilizando uma interface para os diversos bancos de dados envolvidos.

## 5. Referências Bibliográficas

Alpdemir, M. N. et al. (2003), Service-based distributed querying on the grid. In Proc. of the 1st Int. Conf. on Service Oriented Computing. Springer. p. 467-482.

Barbosa, A. C. P.; Porto, F.; Melo, R. N. (2002), Configurable Data Integration Middleware System. Journal of the Brazilian Computer Society. p. 12-19.

Brito, M.; Kakugawa, F. R.; Sato L. M.; et al. (2009), An Architecture for Integrating Databases with Replication Support Based on the OGSA-DAI Middleware. 12th IEEE International Conference on Computational Science and Engineering, CSE'09. Vancouver. p. 298-305.

Brito, M.; Sato, L. M. (2008), Extending OGSA-DAI possibilities with a JDBC driver. 11th IEEE International Conference Computational Science and Engineering, CSE'08. São Paulo. p. 155-162.

Foster, I. (2005), Globus toolkit version 4: Software for Service-Oriented systems. Network And Parallel Computing: IFIP International Conference, Beijing.

Foster, I.; Kesselman, C. (2004), The Grid: Blueprint for a New Computing Infrastructure. San Francisco: Elsevier.

Foster, I.; Kesselman, C.; Tuecke, S. (2001), The Anatomy of the Grid - Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, p. 200-222.

Hong, C. et al. (2007), OGSA-DAI 3.0 - The What's and Whys. Proceedings of the UK e-Science All Hands Meeting, p. 158-165, Setembro.

# Implementação da Interface MPI e de sua Infraestrutura para Grades Computacionais

**Augusto Mendes Gomes Júnior[1], Liria Matsumoto Sato[1], Calebe de Paula Bianchini[2], Francisco Isidro Massetto[3], Nilton Cézar de Paula[4]**

[1]Escola Politécnica – Universidade de São Paulo (POLI-USP)
Caixa Postal 05508-070 – São Paulo – SP – Brasil

[2]Faculdade de Computação e Informática – Universidade Presbiteriana Mackenzie
Caixa Postal 01302-907 – São Paulo – SP – Brasil

[3]Centro de Matemática, Computação e Cognição - Universidade Federal do ABC (UFABC)
Caixa Postal 09210-170 – Santo André – SP – Brasil

[4]Departamento de Computação - Universidade Estadual de mato Grosso do Sul (UEMS)
Caixa Postal 79804-970 – Dourados – MS – Brasil

augustomgjr@usp.br, liria.sato@poli.usp.br,
calebe.bianchini@mackenzie.br,
francisco.massetto@ufabc.edu.br, nilton@comp.uems.br

***Abstract.*** *This paper describes the infra-structure needed to develop the MPI interface to grid computing. In this infer-structure is showed the monitoring, the dispatching and the execution control. Moreover, we detail the MPI primitives developed and the mechanism that make the messages forwarding between different administrative domains.*

***Resumo.*** *Este trabalho descreve a infraestrutura necessária para a implementação da interface MPI para grades computacionais, mostrando como é feito o monitoramento, o disparo e o controle da execução. Além disso, também é detalhado as primitivas MPI implementadas e o mecanismo que faz o encaminhamento de mensagens entre domínios administrativos distintos.*

## 1 INTRODUÇÃO

Em um cenário onde grupos espalhados geograficamente trabalham cooperativamente no desenvolvimento de aplicações com alto nível de processamento, a integração dos seus diferentes recursos computacionais atende esta grande demanda de processamento. Visando este objetivo, grades computacionais (*grid computing*) provêem uma solução apropriada, integrando em uma plataforma, um conjunto de sistemas computacionais.

Para a troca de informações entre os nós pertencentes a um ambiente de grades computacionais (FOSTER et al 1997), é necessária a utilização de uma biblioteca e/ou serviço que ofereça este recurso.

O MPI (MPI2, 2003) é uma interface de passagem de mensagem utilizada, para troca de informações, em *cluster* de computadores. Porém, o MPI não funciona em um ambiente de grades computacionais. O MPIGH2 (KARONIS et al, 2009) é uma implementação

para grades computacionais, mas ele assume que todos os nós têm endereçamento público. Com isso, não há a possibilidade de executar em um *cluster* onde haja nós com endereçamento privado.

O objetivo deste trabalho é a implementação de uma biblioteca de passagem de mensagem MPI, a ser instalada e executada em uma grade computacional. Com a implementação do MPI para este ambiente, um objetivo secundário é o desenvolvimento de uma ferramenta de programação paralela que ofereça a interface MPI. Sendo que esta ferramenta permitirá que a execução de uma aplicação seja distribuída entre os nós da grade, inclusive nós de diversos *clusters*.

A arquitetura da infraestrutura de grade computacional é ilustrada na Figura 1. Toda a interface MPI foi modelada para a arquitetura de grades computacionais, levando-se em consideração as suas peculiaridades, como por exemplo, a máquina de entrada que cada *cluster* possui.

A implementação da arquitetura de comunicação entre processos MPI utilizando os recursos da grade computacional, contempla nós de execução heterogêneos, incluindo *clusters*, computadores SMP e monoprocessados. Uma aplicação paralela pode ser distribuída entre nós de diversos *clusters*, ou ser executada por nós de um único *cluster*.

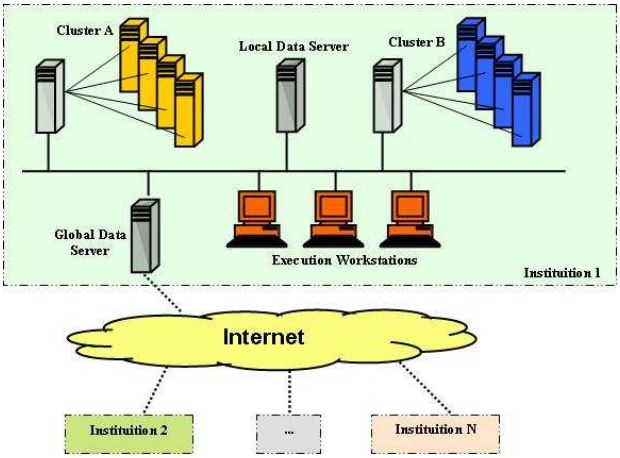



**Figura 1: Arquitetura física da grade computacional.**

As próximas seções detalham a arquitetura do sistema, mostrando a infraestrutura da biblioteca MPI para grades computacionais, a sua modelagem e implementação.

## 2 Arquitetura do Sistema

Para viabilizar a integração, a comunicação, o controle de execução e o gerenciamento dos diversos *sites*/domínios do sistema, faz-se necessária uma infraestrutura que torne o sistema transparente para o usuário. A arquitetura da infraestrutura é apresentada na Figura 2.

Como pode ser observado na Figura 2, a arquitetura é definida em diversos níveis, com funcionalidades específicas:

- *Jobs*: constituem o conjunto de aplicações paralelas que são submetidas à execução no ambiente.
- Implementação MPI: implementação MPI para comunicação dos diversos processos, estejam eles alocados em processadores com máquinas de

endereçamento público ou privados. Como todo acesso e comunicação entre nós internos ao *cluster* e nós externos ao *cluster* devem, obrigatoriamente, passar pelo nó *front-end*, as bibliotecas de comunicação MPI atuais não contemplam esta limitação. Deve-se então, fazer uso de um processo que realize o encaminhamento das mensagens MPI entre nós internos e externos ao *cluster*. O processo denominado *gateway* (MASSETTO et al, 2010) é o responsável por isso, onde cada nó *front-end* deve ter um processo *gateway* instalado para a integração de *clusters*.

- Mecanismo de Escalonamento: utilitários para disparo e controle de execução das aplicações MPI e também para monitoramento dos nós que compõem o sistema. O mecanismo de escalonamento utilizado foi o GCSE e o LIMA (PAULA, 2009). O escalonador GCSE provê recursos apropriados para a execução de aplicações paralelas com a distribuição de processos MPI para múltiplos nós da grade computacional que podem ser *clusters* ou servidores. Para obter um bom desempenho, é necessário obter informações sobre os diversos nós, incluindo capacidade de processamento, hardware e carga de trabalho atualizada. O sistema de monitoramento LIMA provê estas informações e é utilizado pelo GCSE.
- Globus: o Globus (FOSTER, 2010) é uma infraestrutura de integração de grades computacionais que oferece, dentre outras funcionalidades, recursos para alocação de nós e autenticação entre os diversos componentes do sistema (uma vez que pode haver vários usuários distintos com credenciais distintas). Todos os nós do sistema com endereçamento público devem ter o Globus instalado.
- Condor: responsável pelo escalonamento, disparo e controle de execução em cada ambiente local. O Condor (CONDOR, 2010) provê um mecanismo de fila, uma política de escalonamento com ou sem prioridades, e um esquema de classificação de recursos. Para a utilização do Condor, o usuário submete seu *job* ao Condor, que o coloca em uma fila, o executa, e depois informa ao usuário o resultado final.

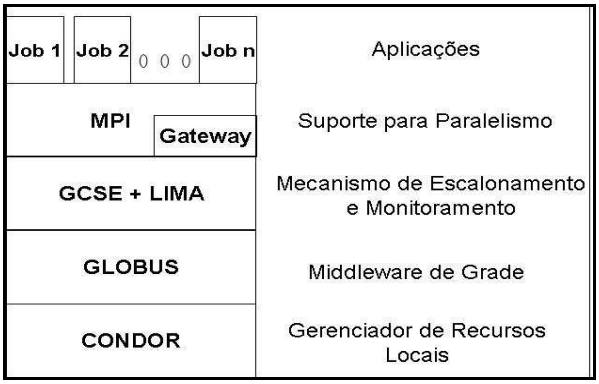



**Figura 2: Arquitetura do sistema.**

## 3 Implementação do MPI para Grades Computacionais

Como o objetivo deste trabalho é a utilização do MPI para grades computacionais, foi necessária toda a sua modelagem para este tipo de ambiente. Para a modelagem e implementação, as primitivas MPI foram divididas em categorias, de acordo com a sua funcionalidade, sendo as seguintes categorias listadas, e consequentemente, as seguintes

primitivas foram implementadas: controle de ambiente (*MPI_Init*, *MPI_Init_thread* e *MPI_Finalize*); manipulação do comunicador (*MPI_Comm_rank* e *MPI_Comm_size*); comunicação ponto-a-ponto (*MPI_Recv*, *MPI_Irecv*, *MPI_Send*, *MPI_Isend*, *MPI_Bsend*, *MPI_Ssend*, *MPI_Test* e *MPI_Wait*); comunicação coletiva (*MPI_Barrier*, *MPI_Bcast*, *MPI_Reduce*, *MPI_Scatter* e *MPI_Gather*); manipulação do buffer (*MPI_Pack*, *MPI_Unpack*, *MPI_Pack_size*, *MPI_Buffer_attach* e *MPI_Buffer_detach*); e utilitárias (*MPI_Wtime*, *MPI_Get_count*, *MPI_Get_processor_name*, *MPI_Get_elements* e *MPI_Probe*).

O ambiente de desenvolvimento é composto por nós e domínios administrativos distintos. Quando o usuário solicitar o disparo da aplicação, o escalonador vai retornar a lista de nós disponíveis para a execução. A ferramenta MPI implementada possui a mesma interface que as demais implementações MPI. Isso a torna totalmente portável para qualquer ambiente. Para a implementação das primitivas, foram utilizadas estratégias para minimizar a comunicação entre os processos, especialmente nas primitivas coletivas.

Durante todo o desenvolvimento das primitivas de comunicação, um ponto fundamental foi o desenvolvimento da ferramenta de encaminhamento de mensagens (*gateway*). O *gateway* foi desenvolvido (MASSETTO et al, 2010) para suportar múltiplas requisições de envios e recebimentos.

## 4 Conclusão

Atualmente, a ferramenta MPI para grades computacionais está desenvolvida, assim como o *gateway*. Os testes funcionais foram realizados para validar a ferramenta, e agora estão sendo realizados testes de desempenho, onde a ferramenta implementada será comparada com implementações MPI nativas. Espera-se que ao final dos testes, a ferramenta seja disponibilizada, em uma infraestrutura de grade computacional, para que usuários possam fazer submissões de aplicações MPI para grades.

## Referências

CONDOR. Projeto Condor. Disponível em http://www.cs.wisc.edu/condor/ . Acesso em 03/2010.

FOSTER, I. e KESSELMAN, C.. Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications and High Performance Computing, 11(2):115-128, Summer 1997.

FOSTER, I. A Globus Primer. Draft Paper. Disponível em http://www.globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf. Acesso em 03/2010.

KARONIS, N. et al. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Disponível em www.globus.org . Acesso em 09/2009.

MASSETTO, F. I. et al. A Message Forward Tool for integration of Clusters of Clusters based on MPI Architecture. The 2nd Russia-Taiwan Symposium on Methods and Tools of Parallel Programming Multicomputers. Russia, Maio, 2010.

MPI2, MPI-2 : The Message-Passing Interface. University of Tennessee, 2003.

PAULA, N. C. Um ambiente de monitoramento de recursos e escalonamento cooperativo de aplicações paralelas em grades computacionais. Tese de doutorado: POLI/USP, 2009.

# Uma Abordagem Orientada a Sistemas para Otimização de Escalonamento de Processos em Grades Computacionais

**Paulo H. R. Gabriel, Rodrigo F. Mello**

[1]Instituto de Ciências Matemáticas e de Computação – USP
Avenida Trabalhador são-carlense, 400 – Centro
Caixa Postal: 668 – CEP: 13560-970 – São Carlos/SP

{phrg,mello}@icmc.usp.br

***Resumo.*** *Grades computacionais são ambientes para compartilhamento de recursos heterogêneos e distribuídos, os quais podem ser utilizados de maneira cooperativa para resolver problemas de alta demanda computacional. Esses aspectos motivaram a adoção de grades, no entanto, outros problemas surgiram com a implementação desses ambientes, tais como a proposta de padrões para protocolos de comunicação, aspectos relacionados à segurança, consistência de dados e escalonamento de processos. No contexto de escalonamento de processos, diversas políticas foram propostas, as quais tipicamente consideram a demanda de aplicações por recursos computacionais. Em geral, esses trabalhos visam reduzir o tempo de execução de tarefas. Essas abordagens, denominadas orientadas a aplicações, não avaliam, contudo, a eficiência de utilização e rendimento dos recursos disponíveis. Essa limitação motivou novos estudos no sentido de propor políticas orientadas a recursos, as quais, por sua vez, avaliam a subtilização dos mesmos. Recentemente, alguns trabalhos sugerem a combinação de ambas abordagens, dando origem às políticas orientadas a sistemas. Essas políticas, atualmente, consideram combinações e simplificações desses objetivos conflitantes, não os atendendo simultaneamente. Essa restrição motiva este trabalho a propor uma abordagem orientada a sistemas e multiobjetivo que avalia, simultaneamente, objetivos de técnicas orientadas a aplicações e a recursos por meio de suas dinâmicas comportamentais. A abordagem proposta considera séries temporais para descrever a dinâmica comportamental de processos e recursos, as quais serão utilizadas para caracterizar tendências e situações futuras, de modo a encontrar ótimos globais estáveis e, consequentemente, contribuir para o aumento de desempenho de execução de aplicações sem, contudo, subutilizar recursos.*

## 1. Introdução

Grades computacionais são ambientes para compartilhamento de recursos heterogêneos e distribuídos, os quais podem ser utilizados de maneira cooperativa para resolver problemas de alta demanda computacional [Foster et al. 2001]. A concepção de ambientes de grade, no entanto, introduziu novos problemas relacionados ao uso eficiente de recursos distribuídos, considerando aspectos relativos a proposta de padrões para protocolos de comunicação, consistência de dados e escalonamento de processos.

O escalonamento de processos envolve a alocação de tarefas sobre recursos disponíveis no ambiente distribuído. Essa atividade deve considerar a capacidade e a carga

instantânea de recursos e as demandas de tarefas. Essas demandas envolvem cargas de processamento, memória, acesso a disco e comunicação em rede. Além disso, tarefas podem apresentar diferentes graus de dependência entre si, o que influencia, também, na abordagem de escalonamento adotada. Não obstante a complexidade envolvida nesses aspectos, as capacidades de uma grade, as cargas de recursos e ocupações de processos são dinâmicas, ou seja, variam no decorrer das observações.

Essa variabilidade tem limitado diversos trabalhos na área de grades computacionais, os quais, usualmente, consideram escalonamento estático de recursos e aplicações do tipo *Bag-of-Tasks* (essas aplicações são compostas por tarefas independentes, com pouca ou nenhuma comunicação entre si) [Andrade et al. 2003, Foster 2005, Costa et al. 2003, Goldchleger et al. 2004]. Muitas dessas abordagens consideram o escalonamento de aplicações com modelos de interação previamente conhecidos, os quais são, geralmente, descritos por grafos direcionados [Kasahara and Tobita 2002].

Além disso, diversas políticas propostas consideram, tipicamente, a demanda de aplicações por recursos computacionais. Em geral, esses trabalhos visam reduzir o tempo de execução de tarefas. Essas abordagens, denominadas orientadas a aplicações, não avaliam, contudo, a eficiência de utilização e rendimento dos recursos disponíveis. Essa limitação motivou novos estudos no sentido de propor políticas orientadas a recursos, as quais, por sua vez, avaliam a subutilização dos mesmos [Dong and Akl 2006]. Recentemente, alguns trabalhos sugerem a combinação de ambas abordagens, dando origem às políticas orientadas a sistemas [Li and Li 2009], a qual emprega uma combinação de múltiplos objetivos, visando auxiliar tanto o desempenho das aplicações quanto a ocupação uniforme do ambiente. Essas políticas, atualmente, consideram combinações e simplificações desses objetivos conflitantes, não os atendendo simultaneamente.

## 2. Proposta

Este trabalho propõe um modelo de otimização multiobjetivo de escalonamento de processos orientado a sistemas, o qual emprega dinâmicas comportamentais de processos e recursos. Nessa abordagem, séries de comportamentos individuais de processos e recursos (tais como, observações temporais de consumo de processamento, acessos a memória, disco rígido e rede) serão estudadas com o objetivo de compreender suas regras geradoras. A partir dessas regras pode-se modelar relações de dependência entre observações e, consequentemente, predizer operações futuras. Essas regras geradoras bem como as predições individuais serão cruzadas a fim encontrar pontos estáveis entre múltiplas séries (isto é, atingir múltiplos objetivos de otimização) sob diferentes dinâmicas. Esses pontos indicam situações onde os múltiplos objetivos dessas séries são atendidos.

Considere, por exemplo, um processo $p_1$ cujo consumo de CPU é linear em relação ao tempo (curva vermelha do gráfico da Figura 1). Considere, também, que a oferta de CPU de um determinado computador $m_1$ em relação ao tempo seja dado pela curva azul do gráfico da Figura 1. Assim, quando o processo $p_1$ é alocado no computador $m_1$, tem-se a situação ilustrada pelo gráfico da Figura 1. Percebe-se que, a partir de determinado instante, o consumo de CPU por $p_1$ é superior à oferta desse recurso em $m_1$, o que tende a prejudicar o desempenho de tal processo. Por outro lado, até atingir determinado instante, a CPU de $m_1$ está sendo subtilizada.

A Figura 1 ilustra, portanto, dois momentos distintos de políticas de escalona-

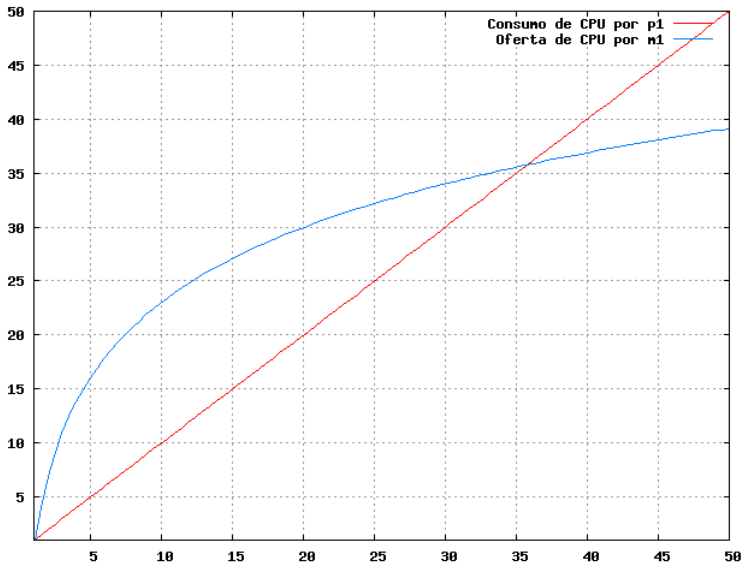



**Figura 1. Exemplo de cenário onde o processo $p_1$ consome CPU do computador $m_1$. Em determinado instante, o consumo excede a oferta do recurso.**

mento: em um primeiro momento, tem-se uma alocação orientada a recursos, uma vez que existem recursos disponíveis para a execução do processo $p_1$; uma política adequada impediria a subutilização desses recursos. Em um segundo momento, no entanto, tem-se um escalonamento orientado a aplicações, visto que os recursos tornam-se insuficientes, sendo necessária uma política que permita maior desempenho na execução do processo. Pode-se observar, portanto, que a utilização individual de cada uma das políticas tende a prejudicar o desempenho de execução de tarefas ou de utilização de recursos.

## 3. Considerações Finais

Com o emprego de otimização multiobjetivo, espera-se encontrar estados estáveis para as séries de processos e recursos presentes na grade computacional, os quais devem contribuir com o aumento no desempenho de execução de processos, sem, contudo, subutilizar recursos. Nesse sentido, a grade pode ser modelada como um sistema dinâmico macroscópico composto por séries individuais que devem tender a estados globais estáveis.

## Agradecimentos

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) – Processo número 2009/15338-1.

## Referências

Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). OurGrid: An approach to easily assemble grids with equitable resource sharing. *Lecture Notes in Computer Science*, 2862:61–86.

Costa, P., Zorzo, S., and Guardia, H. (2003). ProGrid: a proxy-based architecture for grid operation and management. In *Computer Architecture and High Performance Computing, 2003. Proceedings. 15th Symposium on*, pages 100–106.

Dong, F. and Akl, S. G. (2006). Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, Queen's University, Kingston, Ontario.

Foster, I. (2005). Globus toolkit version 4: Software for service-oriented systems. In *Proc. of IFIP International Conference on Network and Parallel Computing*, volume 3779 of *LNCS*, pages 2–13. Springer-Verlag.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. (2004). InteGrade: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice & Experience*, 16(5):449–459.

Kasahara, H. and Tobita, T. (2002). A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5:379–394.

Li, C. and Li, L. (2009). A system-centric scheduling policy for optimizing objectives of application and resource in grid computing. *Computers & Industrial Engineering*. In Press.

# Escalonamento de Workflows com Tarefas Paralelas e Seqüenciais em Grades Computacionais

**Silvio Luiz Stanzani[1], Nilton Cézar de Paula[2], Líria Matsumoto Sato[1]**
[1]Escola politécnica da Universidade de São Paulo (USP)
Avenida Prof. Luciano Gualberto, travessa 3 nº 380 CEP 05508-970 – São Paulo – SP – Brasil
silvio.stanzani@usp.br, liria.sato@poli.usp.br

[2]Universidade Estadual de Mato Grosso do Sul, Cidade Universitária de Dourados - Rod. Dourados/Itahum, Km 12 79804-970 - Dourados, MS - Brasil - Caixa-Postal: 351
nilton.paula@poli.usp.br

***Abstract.*** *This paper describes a framework to provide support for the execution and scheduling of DAGs composed by sequential and parallel tasks, based on strategies considering parallel characteristics of tasks.*

***Resumo.*** *Este artigo descreve um Framework para prover suporte à execução e escalonamento de DAGs composto por tarefas paralelas e seqüenciais, utilizando estratégias que consideram as características do paralelismo das tarefas.*

## 1. Introdução

As Grades Computacionais surgiram a partir da idéia de criar infra-estruturas agregando recursos computacionais geograficamente dispersos e pertencentes a diferentes domínios administrativos, com o objetivo de explorar o poder computacional de computadores e clusters como um único recurso [1]. As arquiteturas de software das Grades Computacionais têm foco na definição de *Middlewares* para abstração da complexidade do ambiente, para prover suporte à programação de aplicações de maneira eficiente.

Os *Middlewares* são fundamentais para a programação de aplicações para Grades Computacionais, no entanto, outros requisitos também influenciam a eficiência do uso desse ambiente, tais como, o gerenciamento de filas de execução de tarefas, execução de workflows, e o escalonamento de tarefas. Os *Resource Brokers (*RBs) tem como objetivo prover suporte a execução de tarefas em ambientes de Grades Computacionais atendendo tais requisitos, com base no uso de *Middlewares*. O uso de estratégias de escalonamento implementadas pelos RBs tem o potencial de melhorar a eficiência do uso dos recursos de uma Grade Computacional considerando diversos aspectos, tais como, falhas no ambiente, custo de utilização de recursos, dependências entre tarefas de um workflow, entre outros.

Os sistemas de programação paralela de baixo nível como o MPI têm sido desenvolvidos para ambientes heterogêneos, permitindo utilizar diversos *clusters* para a execução de aplicações paralelas. Uma parte desses sistemas utiliza o suporte de *Middlewares* de Grade Computacional [12]. A execução de aplicações paralelas utilizando tais sistemas envolve requisitos, tais como, execução da aplicação em caso de

falha de recursos, escolha de recursos com menor latência de rede, entre outros. Nesse contexto, o uso de estratégias de execução de workflows e o uso de estratégias de escalonamento levando em consideração as características de paralelismo das aplicações têm o potencial de atender tais requisitos, e prover suporte à execução de aplicações paralelas em ambientes de Grades Computacionais de maneira eficiente. Esse trabalho tem como objetivo desenvolver um Framework de execução e escalonamento de Workflows com tarefas paralelas e seqüenciais em ambientes de grades computacionais.

### 1.1. Motivações

As principais motivações para esse trabalho partiram do envolvimento do LAHPC em dois projetos: O projeto OpenModeller e o projeto Rede Galileu. No contexto do projeto OpenModeller [6] foi desenvolvida uma ferramenta para prover suporte ao processo de modelagem de nicho ecológico. Tal processo é computacionalmente intensivo e foi necessário explorar estratégias de escalonamento de aplicações paralelizadas com MPI com suporte do Condor [7]. O projeto Rede Galileu tem como objetivo criar uma rede colaborativa para suporte a execução de aplicações computacionalmente intensiva da área de engenharia naval. Nesse projeto está sendo criado um ambiente de Grade Computacional para interligar *clusters*, e um *Framework* para prover suporte à execução de aplicações em ambientes multi-cluster.

## 2. Escalonamento de Tarefas em Grades Computacionais

Em um ambiente de grade computacional uma aplicação é especificada como uma tarefa, que é composta pelo executável da aplicação, arquivos de entrada e saída, além dos recursos necessários à execução. Os recursos podem incluir bibliotecas de software, fontes de dados, arquivos, bem como, requisitos de hardware, tais como, uma arquitetura específica de computador, uma quantidade mínima de memória, entre outros. O escalonamento de tarefas é um processo iniciado a partir de um conjunto de tarefas e consiste em escolher os recursos necessários à execução das tarefas a partir de um conjunto de recursos disponíveis [3] [4]. O escalonamento de tarefas pode ser feito considerando um conjunto de tarefas independentes ou considerando um conjunto de tarefas com dependências entre si, que são definidos como um DAG (*Directed Acyclic Graph*). Diversas estratégias para escalonamento de DAGs foram criadas [5], os RBs implementam tais estratégias para otimizar diferentes aspectos da execução de aplicações. Na seção a seguir é mostrado o *framework* que está em desenvolvimento para executar e escalonar DAG utilizando estratégias que considerem os aspectos de paralelismo das aplicações.

## 3. Um Framework para Execução de DAGs com Tarefas Paralelas e Seqüenciais em Grades Computacionais

O objetivo desse trabalho é prover suporte a execução e escalonamento de DAGs com tarefas seqüenciais ou desenvolvidas com suporte de *Frameworks* de paralelismo em grades computacionais. Para atender esse objetivo um *Framework* foi desenvolvido com base no GCSE (*Grid Cooperative Scheduling Environment*) [11], para executar tarefas paralelas em ambientes de Grades Computacionais [1].

O *Framework* é composto por dois componentes. O primeiro componente chamado **WFMSMgr** é responsável pela execução do DAG. O segundo componente chamado **WFMSScheduler é** responsável por escalonar as tarefas do DAG. A estratégia de escalonamento é baseada nas características do paralelismo, tais como, uso de *Threads* ou *OpenMP*, quantidade de nós que pode ser usada, e informações sobre o uso de arquivos. O componente **WFMSMgr** realiza a monitoração dos recursos da grade computacional por meio do MDS e executa as tarefas utilizando GRAM ou o GCSE. O componente **WFMSScheduler** escolhe os nós de execução para as tarefas dos DAGs esperando para serem executadas. Um banco de dados armazena os dados sobre as tarefas e sobre os recursos disponíveis, dessa forma, é possível alterar o escalonamento de maneira dinâmica, bem como, integrar diferentes estratégias de escalonamento ao componente **WFMSScheduler.** O DAG é especificado segundo a linguagem DAGMan [7] e cada tarefa do DAG é especificada de acordo com a linguagem JSDL (*Job Submission Description Language*) [13].

A Figura 1 mostra a seqüência completa de execução de um DAG. O processo inicia com o usuário que submete um DAG, em seguida o componente **WFMSMgr** armazena as tarefas do DAG no banco de dados, depois disso o **WFMSScheduler** obtém as tarefas prontas para escalonamento e em seguida armazena a tarefa escalonada, o último passo é feito pelo componente **WFMSMgr** que obtem as tarefas escalonadas e as submete para execução, usando o GCSE [11].

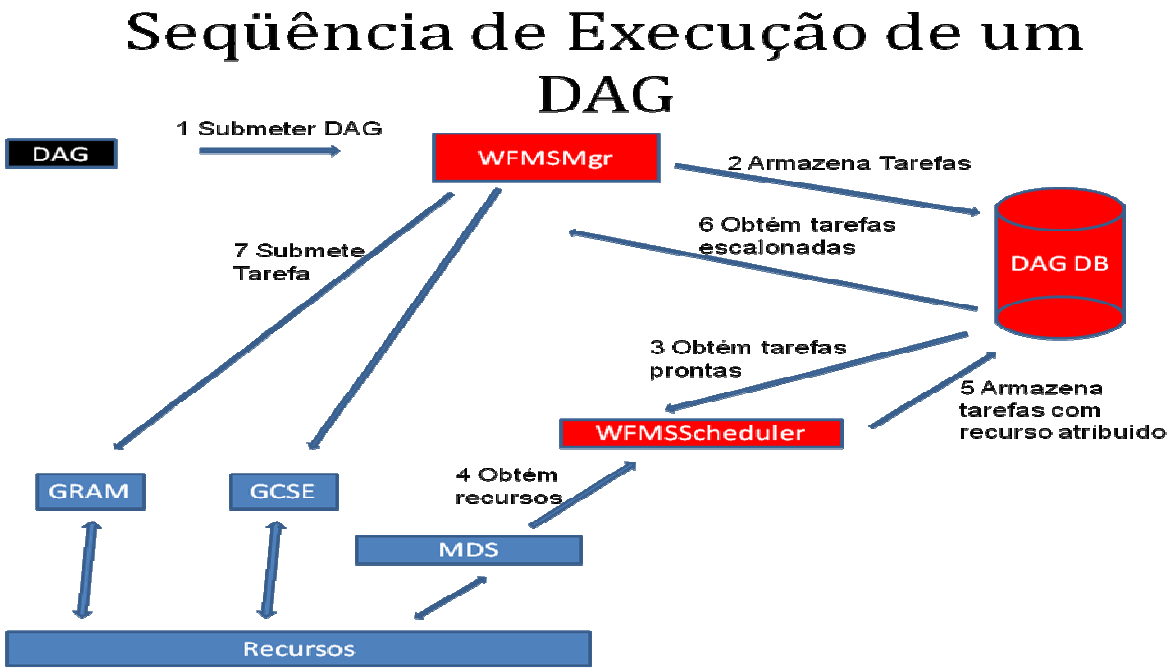



**Figura 1. Seqüência de execução de um DAG.**

## 4. Discussão e Conclusão

Nesse artigo foi descrita a proposta de um *framework* para o escalonamento de DAGs composto por tarefas seqüenciais e paralelas. O componente de escalonamento desse *Framework* permitirá desenvolver estratégias para a execução de tarefas paralelas em

ambientes de grades computacionais, bem como, avaliar modelos de paralelismo adequados de serem utilizados por aplicações para ambientes de grades computacionais.

## 5. Referências


[1] I.T. Foster, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, Springer-Verlag, 2001, pp. 1-4.

[2] J. Yu e R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Rec.*, vol. 34, 2005, pp. 44-49.

[3] R. Buyya, *High Performance Cluster Computing: Architectures and Systems, Vol. 1*, Prentice Hall, 1999.

[4] J.M. Schopf, "Ten actions when Grid scheduling: the user as a Grid scheduler," *Grid resource management: state of the art and future trends*, Kluwer Academic Publishers, 2004, pp. 15-23.

[5] J. Yu, R. Buyya, e K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing.", Metaheuristics for Scheduling in Distributed Computing Environments, F. Xhafa and A. Abraham (eds), ISBN: 978-3-540-69260-7, Springer, Berlin, Germany, 2008.

[6] M. de Souza Muñoz, R. De Giovanni, M. de Siqueira, T. Sutton, P. Brewer, R. Pereira, D. Canhos, e V. Canhos, "openModeller: a generic approach to species' potential distribution modelling," GeoInformatica, 2009.

[7] D. Thain, T. Tannenbaum, e M. Livny, "Distributed computing in practice: the Condor experience: Research Articles," Concurr. Comput. : Pract. Exper., vol. 17, 2005, pp. 323-356.

[8] T. Fahringer, R. Prodan, Rubing Duan, F. Nerieri, S. Podlipnig, Jun Qin, M. Siddiqui, Hong-Linh Truong, A. Villazon, e M. Wieczorek, "ASKALON: A Grid Application Development and Computing Environment," The 6th IEEE/ACM International Workshop on Grid Computing, 2005., Seattle, WA, USA: , pp. 122-131.

[9] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, e D.S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," Sci. Program., vol. 13, 2005, pp. 219-237.

[10] J. Yu e R. Buyya, "Gridbus Workflow Enactment Engine." Grid Computing: Infrastructure, Service, and Applications, 119-146pp, L. Wang, W. Jie, and J. Chen (eds), ISBN: 978-1420067668, CRC Press, Boca Raton, FL, USA, April 2009.

[11] N. C. de Paula. "Um Ambiente de Monitoramento de Recursos e Escalonamento Cooperativo de Aplicações Paralelas em Grades Computacionais". Escola Politécnica da Universidade de São Paulo. São Paulo 2009.

[12] Karonis, N. T., Toonen, B., and Foster, I. 2003. MPICH-G2: a Grid-enabled implementation of the Message Passing Interface. J. Parallel Distrib. Comput. 63, 5 (May. 2003), 551-563. DOI= http://dx.doi.org/10.1016/S0743-7315(03)00002-9

[13] "Job Submission Description Language (JSDL) Specification, Version 1.0". Global Grid Forum. 2005. http://www.gridforum.org/documents/GFD.56.pdf.