Integração de Ganglia, libRastro e Pajé para o Monitoramento de Aplicações Paralelas*

Marcelo Veiga Neves
veiga@inf.ufsm.br

Tiago Scheid
scheid@inf.ufsm.br

Lucas Mello Schnorr
schnorr@inf.ufsm.br

Andréa Schwertner Charão
andrea@inf.ufsm.br

Curso de Ciência da Computação – Universidade Federal de Santa Maria
Campus UFSM – 97105-900 – Santa Maria – RS – Brasil

Resumo

Este artigo trata do uso integrado de diferentes ferramentas de monitoramento a fim de aprimorar a capacidade de análise das execuções de aplicações paralelas. Em particular, descreve-se o processo de integração dos dados coletados por Ganglia, que é uma ferramenta para monitoramento de clusters*, aos rastros de execução gerados por libRastro, que é uma biblioteca para instrumentação de aplicações paralelas. A visualização dos dados integrados é feita com a ferramenta Pajé. Através de alguns exemplos de visualizações integradas, demonstra-se que as informações sobre o estado do* cluster *complementam os rastros de execução da aplicação, permitindo inclusive detectar eventuais problemas na execução da aplicação.*

1. Introdução

No contexto do desenvolvimento de aplicações para *clusters* de computadores, a análise do comportamento de execuções paralelas pode ser de grande utilidade para a obtenção de alto desempenho. Ao longo do tempo, surgiram várias ferramentas que auxiliam os programadores a analisar as execuções de suas aplicações paralelas. Além disso, as ferramentas de gerenciamento de *clusters* também proliferaram, incorporando facilidades de monitoramento dos recursos do sistema.

Este artigo mostra que a correlação entre dados coletados por diferentes ferramentas de monitoramento pode facilitar a análise do comportamento de aplicações paralelas. Para desenvolver esta abordagem, o presente texto está organizado da seguinte forma: primeiramente, apresenta-se uma breve revisão de literatura sobre monitoramento e visualização no contexto do processamento paralelo (seção 2). Em seguida, descreve-se o processo de integração dos dados coletados pela ferramenta Ganglia aos rastros de execução gerados pela biblioteca libRastro, de modo visualizá-los concomitantemente com ferramenta Pajé (seção 3). Por fim, apresenta-se alguns exemplos de aplicações onde a integração de dados de monitoramento é particularmente vantajosa (seção 4).

2. Ferramentas de Monitoramento e Visualização

O monitoramento de um sistema computacional implica na coleta de dados para posterior análise e visualização. Tratando-se de sistemas paralelos e distribuídos, existe atualmente uma ampla gama de ferramentas de monitoramento disponíveis a programadores, administradores e usuários finais. Ao definir quais dados serão coletados, cada ferramenta trabalha com uma determinada abstração do sistema monitorado. Este artigo concentra-se em dois tipos de ferramentas: aquelas especializadas no monitoramento de *clusters* de computadores, e aquelas que se destinam ao monitoramento de aplicações paralelas e distribuídas.

Entre as ferramentas especializadas no monitoramento de *clusters* de computadores, pode-se citar Ganglia [10], Parmon [4], RVision [8], CIS (*Cluster Information Service*) [1] e ClusterProbe [9]. Em geral, estas ferramentas coletam, de maneira integrada, informações sobre a utilização dos recursos de um ou mais *clusters*. Isto inclui, por exemplo, a utilização dos processadores, memórias e dispositivos de entrada e saída de cada nó, bem como e utilização da rede de interconexão. Neste nível de abstração, o mo-

* Este trabalho recebeu apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq e da Fundação de Ampararo à Pesquisa do Estado do Rio Grande do Sul - FAPERGS.

nitoramento auxilia grandemente a administração do sistema, permitindo mantê-lo em funcionamento estável e eficiente [2]. De fato, através dos dados coletados, pode-se detectar e corrigir possíveis falhas e gargalos no sistema, bem como realizar um eventual balanceamento de cargas para uma melhor utilização dos componentes e recursos de processamento [11].

Quanto às ferramentas de monitoramento de aplicações, grande parte delas destina-se à instrumentação de programas a fim de gerar rastros de sua execução. Exemplos de ferramentas nesta categoria são libRastro [6] e GRM/PROVE [3]. Geralmente, os eventos rastreados incluem o início e o término de processos e *threads*, o envio e a recepção de mensagens, bem como outros eventos definidos pelo usuário, específicos a cada aplicação. O monitoramento neste nível de abstração permite que os desenvolvedores ou usuários finais acompanhem a execução de seus programas, identificando eventuais problemas de sincronização, comunicação e desempenho.

Independentemente do nível de observação escolhido, os dados coletados por uma ferramenta de monitoramento são mais facilmente interpretados com o auxílio de uma ferramenta de visualização. Em muitos casos, a própria ferramenta de monitoramento inclui uma interface especializada para observação dos dados coletados. Este é o caso, por exemplo, de Ganglia, que fornece uma interface Web para visualização dos recursos monitorados Existem, também, ferramentas de visualização mais genéricas, que se prestam à visualização de dados de monitoramento provenientes de várias ferramentas e em diferentes níveis de abstração. Este é o caso, por exemplo, de Pablo[12] e Pajé[5].

O restante desta seção apresenta em maiores detalhes as ferramentas utilizadas no contexto deste trabalho: Ganglia, para monitoramento de clusters; libRastro, para monitoramento de eventos em aplicações paralelas, e Pajé, para visualização de execuções paralelas e distribuídas.

2.1. Monitoramento de *Clusters* com Ganglia

Ganglia [10] é um sistema de monitoramento distribuído e escalável para sistemas de computação de alto desempenho, como *clusters* e *grids*. Este sistema é baseado em uma arquitetura hierárquica focada em federações de *clusters*. A implementação de Ganglia é robusta e já foi portada para vários sistemas operacionais e arquiteturas, de modo que esta ferramenta é atualmente utilizada em um grande número de *clusters* em todo o mundo.

Ganglia monitora dois tipos de métricas: as pré-definidas e as definidas pelo usuário. O primeiro tipo é formado pelas métricas coletadas pelo próprio sistema, como por exemplo porcentagem de uso de CPU, carga média, uso de memória, rede, etc. Já as métricas definidas pelo usuário são informações arbitrárias coletadas por programas externos e incorporadas em Ganglia através de uma interface específica. Esta característica faz de Ganglia uma ferramenta extensível, que pode ser utilizada para monitorar qualquer informação possível de ser coletada.

Uma importante característica de Ganglia é sua escalabilidade associada a uma baixa sobrecarga. Para garantir isso, Ganglia utiliza limites (*thresholds*) e intervalos de tempo aleatórios para difusão das métricas coletadas em cada nó.

Para armazenamento e visualização dos dados monitorados, Ganglia utiliza o sistema RRDtool (*Round Robin Database*). Este sistema permite o armazenamento de seqüências temporais de dados de forma compacta em um banco de dados circular, de tamanho constante. A partir dos dados armazenados, RRDtool gera gráficos que mostram a evolução de uma ou mais métricas ao longo do tempo.

A figura 1 ilustra a interface Web de Ganglia para visualização das métricas monitoradas. Nesta interface, observa-se na parte superior um menu de opções onde se pode selecionar a métrica e a granularidade de tempo a ser visualizada. Já na parte central estão as métricas globais do *cluster* e, na parte inferior, a visualização dos estados de cada nó com relação a uma métrica escolhida no menu superior.

Figura 1. *Front-end* Web para visualização de métricas monitoradas com Ganglia.

2.2. Monitoramento de Aplicações com libRastro

A biblioteca libRastro [6] permite monitorar uma aplicação paralela através do registro de eventos observados durante a sua execução, como por exemplo o envio ou recepção de mensagens, a criação de *threads*, etc. Em uma plataforma do tipo *cluster*, os eventos são registrados em cada nó para, após o término da execução, serem sincronizados e combinados em um único arquivo de rastros. Uma característica importante de libRastro é sua genericidade, isto é, sua total independência em relação à aplicação que a utiliza, à biblioteca de comunicação empregada e à ferramenta de visualização utilizada.

De fato, ao contrário de muitos geradores de rastros atrelados a uma ferramenta de comunicação ou visualização, libRastro não atribui semântica aos dados no momento do seu registro ou leitura. O conhecimento da semântica dos dados monitorados fica, assim, sob responsabilidade dos programas que utilizam esta biblioteca. Além disso, com o auxílio de um conversor dos arquivos gerados por libRastro, pode-se produzir rastros no formato de qualquer ferramenta de visualização. Trata-se, portanto, de uma biblioteca versátil, que já foi utilizada, por exemplo, para monitorar aplicações Java distribuídas [7].

2.3. Visualização de Aplicações Paralelas com Pajé

Pajé [5] é uma ferramenta de visualização que permite observar o comportamento de aplicações em ambientes de execução paralelos e distribuídos. Para isso, Pajé utiliza rastros gerados por outras ferramentas durante a execução das aplicações.

Uma particularidade de Pajé é a combinação de três propriedades importantes no contexto da visualização de aplicações paralelas: interatividade, escalabilidade e extensibilidade. A interatividade de Pajé é garantida por uma interface gráfica que mostra diferentes eventos em um diagrama espaço-tempo, permitindo alterar a escala de visualização, avançar e voltar no tempo e, também, inspecionar os detalhes de cada evento. A escalabilidade, por outro lado, refere-se à habilidade de lidar com um grande número de objetos visuais (por exemplo, aplicações com muitas *threads*). Já a extensibilidade é garantida pela possibilidade de visualizar informações provenientes de diferentes ferramentas de monitoramento, em diferentes níveis de abstração.

Para a visualização dos eventos de um rastro, Pajé organiza hierarquicamente os objetos a serem mostrados. Esta hierarquia é definida no arquivo de entrada lido por Pajé, podendo representar, por exemplo, um *cluster* com seus nós, cada nó com seus processos, cada processo com suas *threads*, e assim por diante. A figura 2 mostra um exemplo de visualização com Pajé. Neste exemplo, uma aplicação executa em um *cluster* com 4 nós, sendo que cada nó executa um único processo. Esta hierarquia fica visível na parte esquerda da figura. A execução em si aparece no centro da figura (diagrama espaço-tempo), onde os retângulos representam estados dos processos e as flechas representam comunicações entre eles. Nota-se que diferentes estados de um processo (por exemplo, bloqueado ou executando) são representados por retângulos em tons diferentes.

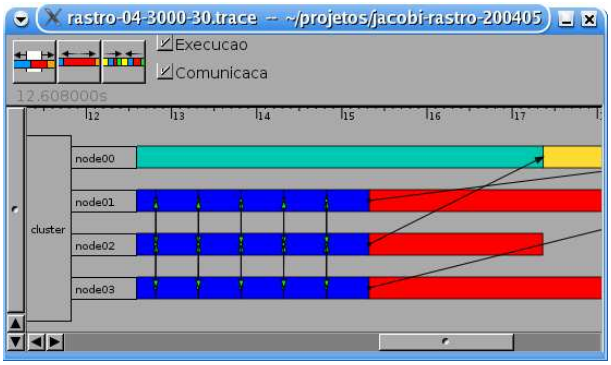

Figura 2. Visualização de uma aplicação paralela com Pajé.

3. Integração de Ferramentas de Monitoramento

Como discutido na seção anterior, o monitoramento dos recursos de um sistema computacional pode ser feito em diferentes níveis de abstração. De fato, utilizando-se ferramentas distintas, pode-se monitorar desde o *hardware* até a lógica da aplicação. No monitoramento de aplicações paralelas, os dados coletados por diferentes ferramentas, inclusive em outros níveis de abstração, podem se complementar e proporcionar ao desenvolvedor uma visão mais completa da execução de seus programas. No entanto, a correlação manual de informações provenientes de diferentes ferramentas pode ser uma tarefa trabalhosa.

Uma solução para este problema é a integração automática dos dados de monitoramento coletados por diferentes ferramentas, de modo a sincronizá-los e observá-los concomitantemente por meio de uma única ferramenta de visualização. O processo de integração depende fortemente das características de cada ferramenta utilizada.

A figura 3 ilustra o processo de integração dos dados monitorados por Ganglia aos eventos registados por libRastro. Como mencionado anteriormente, Ganglia coleta dados em cada nó, sendo que o histórico do estado do *cluster* fica armazenado em uma base de dados RRD (*Round Robin Database*). Já libRastro gera vários arquivos, contendo os even-

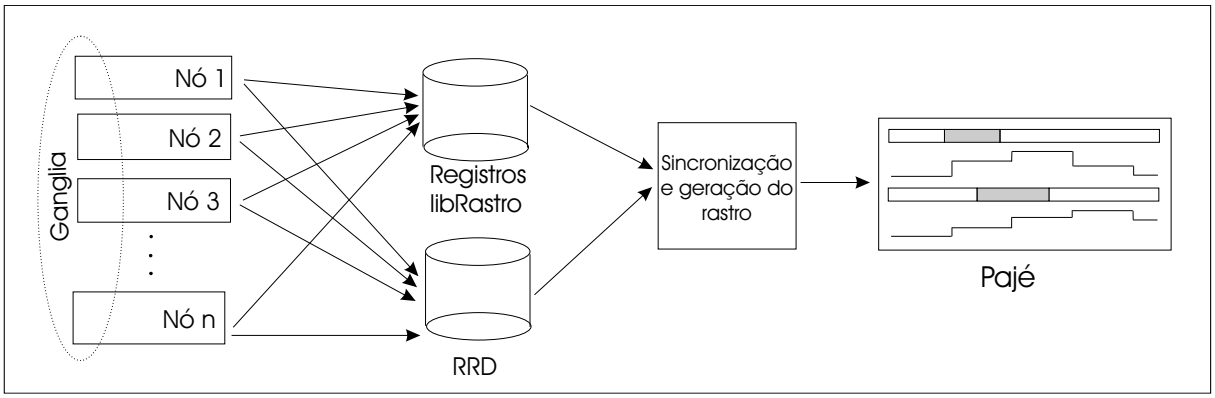

Figura 3. Processo de Integração.

tos registrados, um para cada nó ou *thread*. Assim é necessário que as informações provenientes dessas duas fontes de dados sejam sincronizadas a fim de gerar um único rastro, já que Pajé ainda não trabalha com mais de uma fonte de dados.

No restante desta seção, descreve-se a implementação da integração dos dados coletados por Ganglia aos rastros de execução gerados por libRastro, a fim de visualizá-los com Pajé.

Inicialmente, foi necessário implementar a leitura dos dados registrados por Ganglia. O fato de Ganglia armazenar os dados monitorados em uma base de dados RRD facilita a leitura, pois basta ler uma única fonte de dados para se obter as informações de todo o *cluster*. Por outro lado, estas informações estão armazenadas em um formato próprio, pouco documentado, que teve de ser estudado em detalhes para sua posterior conversão.

Em um segundo momento, foi preciso trabalhar com os dados gerados por libRastro. Esta biblioteca registra os eventos gerados durante a execução de uma aplicação em um formato compacto, assim é necessário decodificar estes arquivos para gerar um rastro intermediário que possa ser facilmente convertido para o formato Pajé.

O passo seguinte foi a integração das informações geradas por libRastro e Ganglia em um único rastro visualizável com Pajé. Houve a preocupação de obter uma forma visualização que facilitasse a correlação dos dados. Para isso preferiu-se representar as duas fontes de dados em um mesmo nível na hierarquia, dado o fato de que o formato de entrada da ferramenta Pajé permite representar dados de forma hierárquica. Assim, pode-se visualizar, por exemplo, as informações coletadas por Ganglia e libRastro referentes a um mesmo nó do *cluster*.

O principal problema enfrentado para a integração dos rastros girou em torno da sincronização das fontes de dados, de modo a tornar a visualização mais significativa. Em particular, foi necessário diminuir os intervalos e a aleatoriedade na difusão dos dados monitorados com Ganglia. Para não comprometer a escalabilidade da ferramenta, preferiu-se reduzir apenas o intervalo de coleta das métricas definidas pelo usuário, pois a pequena granularidade só é necessária durante a execução de uma aplicação que se deseja monitorar. Assim, foi necessário implementar um pequeno programa monitor que é lançado junto com a aplicação. Este monitor coleta informações do sistema e as envia para Ganglia, em um intervalo de tempo reduzido, como uma métrica definida pelo usuário.

Por fim, foi necessário implementar a sincronização dos eventos temporais gerados por libRastro com os dados monitorados por Ganglia. Os dados coletados por Ganglia em diferentes nós são sincronizados no momento em que são armazenados em um servidor central. Quanto à libRastro, esta também possui um mecanismo de sincronização de relógios, que ajusta os tempos de todos os eventos com base no relógio de um dos processadores. Com base nisso, o problema foi resolvido sincronizando-se os arquivos gerados por libRastro com base no relógio do servidor de armazenamento de Ganglia. Assim, obteve-se uma sincronização suficiente dada a granularidade dos dados de monitoramento.

4. Monitorando a Execução de uma Aplicação

A fim de avaliar a integração das ferramentas de monitoramento, foi construída uma aplicação que calcula a distribuição de calor em uma placa metálica segundo a equação de Laplace. Para isso, utilizou-se o método de Jacobi, que é um método iterativo para a aproximação por diferenças finitas da referida equação [13].

A paralelização desta aplicação baseou-se na metodologia mestre–escravo. A carga de processamento, representada por uma matriz de NxN pontos definidos sobre a placa metálica, é dividida igualmente entre os processos escravos, sendo que cada escravo processa um conjunto de linhas adjacentes da matriz. O processo mestre aguarda o re-

sultado calculado pelos escravos, recompõe a matriz completa e salva o resultado em um arquivo.

A cada iteração, os escravos calculam novos valores para cada ponto da matriz com base no valor dos pontos vizinhos (esquerda, direita, acima e abaixo). Em seguida, cada escravo se comunica com os escravos que calculam porções adjacentes da matriz, de modo a atualizar os valores das bordas destas sub-matrizes. Estes valores atualizados são utilizados na iteração seguinte. Este cálculo iterativo prossegue até que o número de iterações desejado seja atingido.

Esta aplicação foi implementada em linguagem C, utilizando a biblioteca MPI para comunicação entre processadores. A instrumentação da aplicação foi feita com libRastro. Em todas as execuções da aplicação, utilizou-se um *cluster* homogêneo composto por 8 máquinas bi-processadas Pentium III a 1 Ghz, com 1 Gbyte de memória RAM, 512 Kbytes de memória *cache* e adaptador de rede Gigabit Ethernet. O sistema operacional de cada máquina é GNU/Linux (distribuição Gentoo 1.4), com *kernel* versão 2.6.2.

Nas próximas seções, apresenta-se e discute-se duas visualizações da execução desta aplicação. Na primeira, os dados de monitoramento da aplicação, coletados com libRastro, aparecem correlacionados com a informação coletada por Ganglia sobre o uso de CPU no *cluster*. Na segunda visualização, a correlação é feita com dados de monitoramento da rede de interconexão, igualmente coletados por Ganglia.

4.1. Correlação com o Uso de CPU

A figura 4 apresenta a primeira visualização da execução da aplicação. Nesta figura, observa-se à esquerda a hierarquia de objetos: o *cluster*, os nós ("node00" a "node06") e o nível de monitoramento, dividido entre o monitoramento dos eventos da aplicação ("app") e o monitoramento do uso da CPU com Ganglia ("mon"). Ao longo da execução, pode-se observar claramente os momentos em que os processos escravos estão ativos, calculando os novos valores para cada ponto na matriz. Estes estados são representados pelos retângulos escuros no diagrama espaço-tempo. Além disso, pode-se visualizar as comunicações entre os escravos que calculam porções adjacentes da matriz, representadas pelas flechas. Em um determinado instante da execução (ver indicação na figura), nota-se que ocorre um aumento na utilização da CPU em "node02". Isto é provocado, neste caso, pelo lançamento de um outro processo, externo à aplicação, que passa a competir pelo uso da CPU neste nó do *cluster*. Esta competição causa um prolongamento do tempo de cálculo em "node02" (retângulo escuro), provocando um tempo de espera maior (retângulos claros) nos nós vizinhos, que necessitam dos valores das bordas das sub-matrizes.

Também observa-se uma pequena defasagem entre o aumento do uso da CPU e o aparecimento dos bloqueios. Isso se deve ao fato de as informações colhidas em um instante só são publicadas, por Ganglia, no segundo seguinte.

A visualização apresentada na figura 4 ilustra uma situação que pode ocorrer, por exemplo, quando o *cluster* é compartilhado por vários usuários, sem um controle automatizado e rigoroso para alocação exclusiva de recursos. Isto também pode ocorrer devido a algum problema na configuração do *cluster*, permitindo que algum processo "pesado" do sistema seja iniciado enquanto uma aplicação está em execução. Em qualquer destes casos, o desenvolvedor da aplicação observaria um comportamento inesperado ao longo da execução, que seria dificilmente explicado através da simples visualização dos rastros da aplicação.

4.2. Correlação com Informações da Rede

A segunda visualização da execução do programa é apresentada na figura 5. Nesta figura, observa-se a mesma hierarquia de objetos do exemplo anterior, sendo que neste caso aparecem todos os nós do *cluster* ("node00" a "node07") e o nível de monitoramento com Ganglia ("mon") mostra informações sobre o uso da rede. No instante ressaltado na figura, nota-se que existe um aumento da utilização da rede em "node02" e "node03". Isto é causado, neste exemplo, por uma transferência de grande quantidade de dados de "node02" para "node03", por um processo externo à aplicação.

Assim como no exemplo anterior, este evento inesperado e completamente independente da aplicação provoca um prolongamento do estado de cálculo em "node03", de modo que os nós vizinhos passam a bloquear na espera pelos valores das bordas das sub-matrizes. Esse comportamento inesperado não seria facilmente explicado visualizando apenas os rastros gerdados por libRastro.

5. Considerações Finais

O monitoramento da execução de aplicações paralelas em *clusters* de computadores é um assunto que há tempos suscita o interesse da comunidade científica. Existe atualmente uma ampla gama de ferramentas que coletam dados em diferentes níveis de abstração, permitindo monitorar desde o *hardware* até a lógica da aplicação. Ao depurar ou otimizar a execução de suas aplicações, um desenvolvedor ou usuário final pode desejar correlacionar os dados coletados por diferentes ferramentas de monitoramento.

Neste contexto, o presente artigo descreveu o trabalho de integração dos dados de monitoramento gerados por duas ferramentas independentes: Ganglia e libRastro. A primeira se presta principalmente ao monitoramento dos recursos

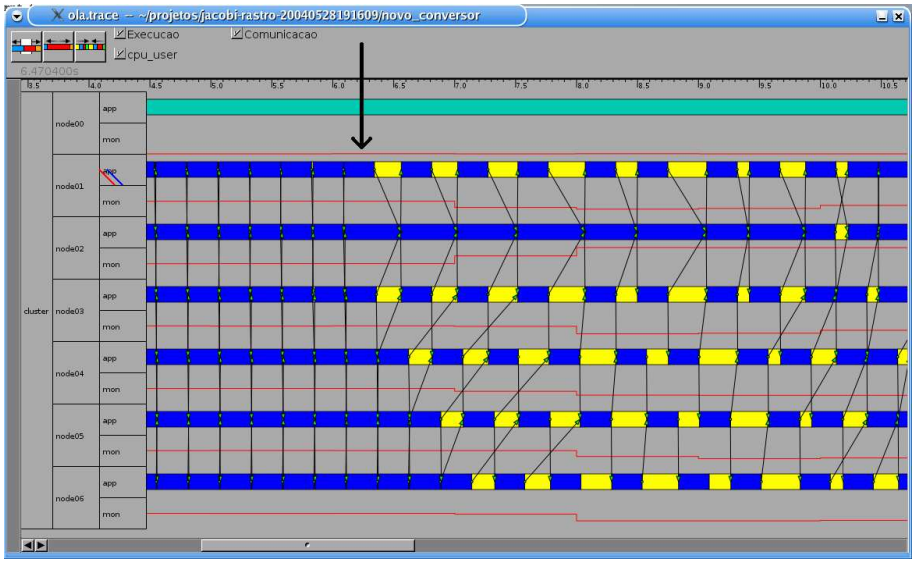

Figura 4. Visualização da correlação com o uso de CPU.

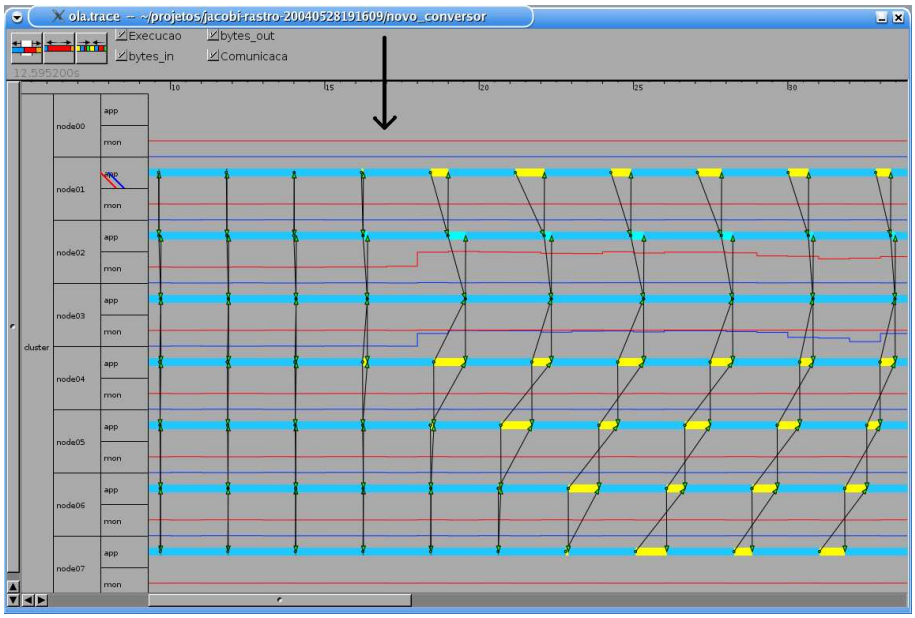

Figura 5. Visualização da correlação com o uso da rede.

de um ou mais *clusters*, enquanto a segunda destina-se à geração de rastros de execução de programas paralelos. Para facilitar a interpretação dos dados coletados independentemente, estes foram combinados em um único rastro visualizável através da ferramenta Pajé.

O uso deste recurso de monitoramento integrado foi ilustrado com diferentes visualizações de uma aplicação paralela simples. Nos exemplos apresentados, foi possível observar que as informações de monitoramento coletadas com Ganglia complementaram o registro dos eventos da aplicação, oferecendo um recurso adicional para análise das execuções paralelas.

Este trabalho está ainda em desenvolvimento, sendo que pretende-se estendê-lo a outras ferramentas de monitoramento. Atualmente, testes estão sendo feitos no núcleo do sistema operacional (no caso, Linux), com o objetivo de

monitorar informações tais como as trocas de contexto de um processo, a comunicação de baixo nível com a placa de rede, entre outras. Acredita-se que estas informações poderão auxiliar no monitoramento de uma aplicação paralela, complementando as informações de monitoramento do *cluster*.

Referências

[1] J. Astalos and L. Hluchý. CIS — A monitoring system for PC clusters. *Lecture Notes in Computer Science*, 1908:225–232, 2000.

[2] M. Baker. Cluster computing white paper. http://dsg.port.ac.uk/ mab/tfcc/WhitePaper/final-paper.pdf, 2000.

[3] Z. Balaton, P. Kacsuk, and N. Podhorszki. Application monitoring in the grid with grm and prove. In *Proc. of the Int. Conf. on Computational Science*, pages 253–262, San Francisco, 2001.

[4] R. Buyya. PARMON: a portable and scalable monitoring system for clusters. *Software Practice and Experience*, 30(7):723–739, June 2000.

[5] J. Chassin de Kergommeaux and B. de Oliveira Stein. Pajé: An extensible environment for visualizing multi-threaded programs executions. *Lecture Notes in Computer Science*, 1900:133–153, 2001.

[6] G. J. da Silva and B. de Oliveira Stein. Uma biblioteca genérica de geração de rastros de execução para visualização de programas. *Anais do I Simpósio de Informática da Região Centro*, 2002.

[7] G. J. da Silva, L. M. Schnorr, and B. de Oliveira Stein. Jrastro: A trace agent for debugging multithreaded and distributed java programs. *15th Symposium on Computer Architecture and High Performance Computing*, page 46, 2003.

[8] T. C. Ferreto, C. A. F. de Rose, and L. de Rose. Rvision: An open and high configurable tool for cluster monitoring. *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, page 75, 2002.

[9] Z. Liang, Y. Sun, and C. Wang. Clusterprobe: An open, flexible and scalable cluster monitoring tool. *International Workshop on Cluster Computing*, pages 261–268, 1999.

[10] M. Massie, B. Chun, and D. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. Technical report, University of California, Berkeley Technical Report, 2003.

[11] M. Pasin and D. Kreutz. Arquiteturas e administração de aglomerados. *Escola Regional de Alto Desempenho*, 3(3):4–34, Jan. 2003.

[12] D. A. Reed and alii. Scalable performance analysis: The Pablo performance analysis environment. *Proceedings of the Scalable Parallel Libraries Conference*, pages 104–113, 1993.

[13] B. Wilkinson and M. Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc., 1999.