UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LUCAS MELLO SCHNORR

Visualização Simultânea e Multi-nível de Informações de Monitoramento de *Cluster*

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Philippe Olivier Alexandre Navaux
Orientador

Prof. Dr. Benhur de Oliveira Stein
Co-orientador

Porto Alegre, maio de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Schnorr, Lucas Mello

Visualização Simultânea e Multi-nível de Informações de Monitoramento de *Cluster* / Lucas Mello Schnorr. – Porto Alegre: PPGC da UFRGS, 2005.

113 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Philippe Olivier Alexandre Navaux; Co-orientador: Benhur de Oliveira Stein.

1. Monitoramento de *cluster*. 2. Depuração de aplicações paralelas. 3. Visualização de informações. 4. Análise multi-nível de dados. 5. Múltiplas fontes de dados. I. Navaux, Philippe Olivier Alexandre. II. Stein, Benhur de Oliveira. III. Título.

“I believe that at every level of society - familial, tribal, national and international - the key to a happier and more successful world is the growth of compassion. We do not need to become religious, nor do we need to believe in an ideology. All that is necessary is for each of us to develop our good human qualities.”

— TENZIN GYATSO, 14TH DALAI LAMA

AGRADECIMENTOS

O trabalho que este texto apresenta é fruto de um desenvolvimento pessoal ao longo dos últimos dez meses, apoiado emocional e intelectualmente por diferentes pessoas e entidades.

Inicialmente, eu gostaria de agradecer a presença e a motivação da minha namorada, Fabiane, que durante todos os dias da execução deste trabalho esteve do meu lado. Gostaria de agradecer também a minha família, mãe e irmã, que me apoiaram diretamente através de um quase completo entendimento dos meus maus humores diários.

Gostaria de agradecer também aos colegas do "aquário": Mano, Márcia, Marlon e Alemão pelas sessões FTP - "Força Tarefa Publicatória", pelas discussões filosóficas, pelas anotações nas paredes forradas de papel pardo e pelo, acredito eu, entendimento dos meus maus momentos. Aproveitando, gostaria de agradecer ao pessoal do Laboratório de Sistemas de Computação da UFSM pelo apoio.

Gostaria também de agradecer ao meu co-orientador, Prof. Benhur, pelos conhecimentos infindáveis, pelas dicas e, claro, eternas correções dos textos. Sua presença acompanhou desde meus primeiros passos na vida acadêmica científica e nas primeiras frases técnicas construídas até a conclusão deste trabalho de mestrado.

Agradeço também ao meu orientador, Prof. Navaux, pelo sorriso motivador e pelas revisões de texto que com certeza melhoraram este texto. Seus problemas de tempo não diminuiram a qualidade de nossas conversas e discussões a respeito do trabalho. Gostaria de agradecer também ao pessoal do Grupo de Processamento Paralelo e Distribuído da UFRGS por todas as vezes que estive em Porto Alegre e fui bem recebido.

Finalmente gostaria de agradecer ao CNPq pelo apoio financeiro, sem o qual este trabalho não teria sido desenvolvido com dedicação exclusiva.

Todas essas pessoas e entidades fazem parte deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

COSMOS	Coblenz Software Monitoring System
CPU	Central Processing Unit
DECK	Distributed Execution and Communication Kernel
FKT	Fast Kernel Tracing
IP	Internet protocol
JVMPI	Java Virtual Machine Profiler Interface
JVMTI	Java Virtual Machine Tool Interface
JVM	Java Virtual Machine
LTT	Linux Trace Toolkit
MPE	MultiProcessing Parallel Environment
MPI	Message Passing Interface
MPICH	Message Passing Interface Portable Implementation
NFS	Network File System
PBS	Portable Batch System
PCP	Performance Co-Pilot
PVM	Parallel Virtual Machine
PMCD	Performance Metrics Collector Daemon
PMDA	Performance Metrics Domain Agent
RMI	Remote Method Invocation
RMS	Real-time Monitoring Subsystem
SCMS	Scalable Cluster Management System
SO	Sistema Operacional
SQL	Structured Query Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VPPB	Visualization and Performance Prediction Tool
XML	Extended Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Clusters de computadores são geralmente utilizados para se obter alto desempenho na execução de aplicações paralelas. Sua utilização tem aumentado significativamente ao longo dos anos e resulta hoje em uma presença de quase 60% entre as 500 máquinas mais rápidas do mundo. Embora a utilização de *clusters* seja bastante difundida, a tarefa de monitoramento de recursos dessas máquinas é considerada complexa. Essa complexidade advém do fato de existirem diferentes configurações de *software* e *hardware* que podem ser caracterizadas como *cluster*. Diferentes configurações acabam por fazer com que o administrador de um *cluster* necessite de mais de uma ferramenta de monitoramento para conseguir obter informações suficientes para uma tomada de decisão acerca de eventuais problemas que possam estar acontecendo no seu *cluster*. Outra situação que demonstra a complexidade da tarefa de monitoramento acontece quando o desenvolvedor de aplicações paralelas necessita de informações relativas ao ambiente de execução da sua aplicação para entender melhor o seu comportamento. A execução de aplicações paralelas em ambientes multi-*cluster* e *grid* juntamente com a necessidade de informações externas à aplicação é outra situação que necessita da tarefa de monitoramento. Em todas essas situações, verifica-se a existência de múltiplas fontes de dados independentes e que podem ter informações relacionadas ou complementares.

O objetivo deste trabalho é propor um modelo de integração de dados que pode se adaptar a diferentes fontes de informação e gerar como resultado informações integradas que sejam passíveis de uma visualização conjunta por alguma ferramenta. Esse modelo é baseado na depuração *offline* de aplicações paralelas e é dividido em duas etapas: a coleta de dados e uma posterior integração das informações. Um protótipo baseado nesse modelo de integração é descrito neste trabalho. Esse protótipo utiliza como fontes de informação as ferramentas de monitoramento de *cluster* Ganglia e Performance Co-Pilot, bibliotecas de rastreamento de aplicações DECK e MPI e uma instrumentação do Sistema operacional Linux para registrar as trocas de contexto de um conjunto de processos. Pajé é a ferramenta escolhida para a visualização integrada das informações. Os resultados do processo de integração de dados pelo protótipo apresentado neste trabalho são caracterizados em três tipos: depuração de aplicações DECK, depuração de aplicações MPI e monitoramento de *cluster*. Ao final do texto, são delineadas algumas conclusões e contribuições desse trabalho, assim como algumas sugestões de trabalhos futuros.

Palavras-chave: Monitoramento de *cluster*, depuração de aplicações paralelas, visualização de informações, análise multi-nível de dados, múltiplas fontes de dados.

Simultaneous and Multi-level Visualization of Cluster Monitoring Information

ABSTRACT

Clusters of computers are commonly used to obtain high performance in the execution of parallel applications. The utilization of clusters has grown through the years and today they are present in almost 60% of the 500 fastest machines in the world. Although the use of clusters is sufficiently spread out, the task of monitoring their resources is considered complex. This complexity comes from the fact that different hardware and software configurations are considered clusters. Beyond that, sometimes the cluster administrator needs more than one monitoring tool to get enough information to understand what might be happening in the cluster. Another situation happens when the information of the execution environment are needed to a better understanding of the parallel applications that are executed. The execution of parallel applications in multi-cluster and grid environments also characterizes the resource monitoring complexity when information of resource utilization needs to be analyzed together with application tracing information. All these situations are characterized by having multiple sources of information that are analyzed independently.

The objective of this work is to propose a data integration model which can adapt itself to different data sources and can provide integrated data to be visualized simultaneously. This model is based on offline parallel application debugging and is divided in two steps: the data tracing and the data integration. A prototype based in this model is described. This prototype uses as data sources the Ganglia and Performance Co-Pilot cluster monitoring tools, with DECK and MPI application tracing, along with an instrumented version of Linux kernel to register the context switches of a set of processes. Pajé was chosen as the visualization tool. The results of the integration process realized by the prototype may be classified among three types: DECK application debugging, MPI application debugging and cluster monitoring. Some conclusions and future works are described in the end of this text.

Keywords: Cluster Monitoring, Parallel Application Debugging, Information Visualization, Multi-level Data Analysis, Multiple Data Sources.

1 INTRODUÇÃO

A utilização de *cluster* de computadores para se atingir alto desempenho de aplicações tem aumentado significativamente ao longo do tempo. Isso se verifica pela análise da listagem das máquinas mais rápidas do mundo (MEUER et al., 2004). Na publicação dessa listagem em novembro de 2004, esse tipo de arquitetura caracteriza quase 60% das máquinas entre as 500 mais rápidas do mundo. A arquitetura de um *cluster* consiste basicamente em um conjunto de computadores independentes e interconectados que juntos representam um único recurso computacional integrado (BUYYA, 1999). Algumas vantagens da utilização desse tipo de arquitetura são a escalabilidade, pela adição de novas máquinas conectadas à rede de interconexão, e o custo, pelo uso de máquinas baratas.

Embora o uso de *clusters* para a obtenção de alto desempenho computacional seja bastante difundida, esse tipo de arquitetura implica na utilização de mecanismos para o monitoramento dos recursos, de forma que o gerenciamento da arquitetura seja realizado. Esse monitoramento consiste basicamente na coleta de informações que caracterizem o estado de cada máquina de um *cluster*. O conjunto de estado das diferentes máquinas compõe o estado global de um *cluster* em um determinado instante. Obter o estado individual das máquinas assim como o estado global de um *cluster* é útil tanto para o administrador desse *cluster*, quanto para programadores e usuários de uma máquina dessa arquitetura.

Embora o monitoramento de *cluster* seja útil de uma maneira geral, essa tarefa é considerada complicada. Isso acontece pelo fato que existem diferentes formas de construir um *cluster* em termos de *hardware*, diferentes combinações de *software* empregadas na configuração das máquinas e diferentes ambientes de execução de aplicações paralelas. Essas características da arquitetura tornam difícil a construção de uma ferramenta de monitoramento que consiga preencher todas as necessidades tanto de administradores quanto de usuários de um *cluster*. Essa dificuldade causou o desenvolvimento de diversas ferramentas de monitoramento, cada uma voltada a uma situação específica.

A existência de variadas ferramentas de monitoramento implica situações onde o administrador do *cluster* é forçado a utilizar mais de uma ferramenta de monitoramento para obter dados suficientes que possam ser utilizadas para uma tomada de decisão. Outras vezes, o programador necessita integrar informações de rastreamento da sua aplicação com dados externos, coletados por uma ferramenta de monitoramento, para melhor entender o comportamento do seu programa (KERGOMMEAUX; VINCENT; OTTOGALLI, 2003). Ainda, quando se utiliza ambientes multi-*clusters* ou *grid* de computadores para a execução de aplicações, é natural encontrar diferentes ambientes com ferramentas de monitoramento possivelmente diferentes. Além dessas situações, grande parte das ferramentas de monitoramento de *cluster* e de depuração de aplicações paralelas são independentes. Essa independência se caracteriza pela forma como são coletados e analisados os dados. A uti-

lização de ferramentas independentes de análise dos dados pode se tornar impraticável em situações onde há a necessidade de se correlacionar as informações coletadas por diferentes ferramentas de monitoramento. Todas essas situações se caracterizam pela existência de múltiplas fontes de informação onde há a necessidade de integrar os dados coletados por elas, de forma que seja possível realizar uma análise dos dados mais abrangente.

O objetivo deste trabalho é propor um modelo para a integração de dados de múltiplas fontes de informação de monitoramento, podendo ser adaptado tanto para a tarefa de monitoramento de *cluster* pelo administrador quanto para a depuração de aplicações paralelas. Dentre os objetivos específicos, pretende-se construir um protótipo baseado no modelo que é proposto e validá-lo através do monitoramento de aplicações e situações de administração de *cluster*. O modelo que é proposto neste trabalho é dividido basicamente em duas etapas: o período de coleta de dados e depois a integração e análise dessas informações. Na proposta do modelo, os dados são coletados através da utilização de ferramentas já existentes de monitoramento de *cluster* e de rastreamento de aplicações paralelas. A segunda parte, de integração dos dados, é feita através da implementação de um protótipo que realiza a sincronização, unificação e padronização dos dados. A sincronização descrita no modelo deve ser feita para manter a relação causal entre os eventos. A unificação dos dados é feita quando um mesmo objeto é identificado de forma diferente por ferramentas de monitoramento diferentes. A padronização implementada no protótipo tem como objetivo converter as informações para um formato único relacionado a uma ferramenta de visualização.

As fontes de informação utilizadas para o protótipo de integração são o Ganglia (SACERDOTI; MASSIE; CULLER, 2003), Performance Co-Pilot (SGI, 1999), rastreamento de aplicações desenvolvidas com a biblioteca DECK e com uma implementação MPI e o rastreamento de processos do Sistema operacional Linux. Pajé (STEIN; KERGOMMEAUX; BERNARD, 2000) é a ferramenta de visualização que será utilizada para a visualização e análise integrada de dados.

O texto que descreve este trabalho está dividido em 6 capítulos e 3 apêndices:

Capítulo 2: Monitoramento de *cluster*

O principal objetivo deste capítulo é apresentar o monitoramento de *cluster* e a sua problemática. Para atingir este objetivo, o capítulo inicialmente apresenta alguns problemas da área de monitoramento de *cluster* e da depuração de aplicações paralelas que podem ser executadas em ambientes como esse. Depois, são apresentadas algumas ferramentas que pretendem resolver esses problemas e que estão relacionadas a este trabalho.

Capítulo 3: Proposta de Modelo Integrado de Monitoramento

Neste capítulo é apresentado o modelo integrado de monitoramento de *cluster* proposto neste trabalho. Inicialmente, são discutidas as características das ferramentas apresentadas no capítulo 2 que levaram à proposta do modelo integrado de monitoramento. Depois, o modelo propriamente dito é apresentado.

Capítulo 4: Implementação do Modelo

Neste capítulo é descrito o protótipo de integração contruído baseado no modelo do capítulo 3. Inicialmente, são descritas a forma como foi feita a coleta de dados de cinco diferentes fontes de informação: Ganglia, Performance Co-Pilot, DECK, MPI e Sistema operacional Linux. Depois, o protótipo propriamente dito é apresentado, descrevendo as decisões de implementação e como foram resolvidas as questões de integração dos dados apresentadas no capítulo 3.

Capítulo 5: Validação

O objetivo deste capítulo é mostrar os resultados obtidos com a integração de dados pelo protótipo. Para isso, diferentes situações de integração de dados são apresentadas. Essas situações são divididas basicamente em três áreas: monitoramento de *cluster*, visualização de aplicações DECK e visualização de aplicações MPI.

Capítulo 6: Conclusão

As principais contribuições do trabalho são revistas neste capítulo. Além disso, são apresentados sugestões de trabalhos futuros e evoluções no modelo proposto e no protótipo desenvolvido.

Apêndice A: Ferramentas

Neste apêndice são descritas as ferramentas e bibliotecas utilizadas neste trabalho. Dentre essas ferramentas estão a biblioteca DECK e a libRastro, assim como as ferramentas Ganglia e Performance Co-Pilot e a ferramenta de visualização Pajé.

Apêndice B: Protótipo de Integração

Neste apêndice são apresentados detalhes da implementação do protótipo. Inicialmente é descrito o processo de adição de novos módulos ao protótipo. No fim, é apresentado um manual de utilização do protótipo.

Apêndice C: Mapeamento de Eventos

Neste apêndice são apresentados os mapeamentos de eventos das cinco fontes de informação utilizadas neste trabalho: DECK, MPI, Ganglia, Sistema Operacional Linux e Performance Co-Pilot. Este mapeamento é utilizado no protótipo para se obter eventos na ferramenta de visualização utilizada neste trabalho, Pajé.

2 MONITORAMENTO DE *CLUSTER*

O monitoramento de *cluster* consiste basicamente na coleta de informações que representem o estado das máquinas desse *cluster*. Normalmente, esse monitoramento é realizado pelo administrador de *cluster* para que ele tenha uma forma de identificar os problemas que possam vir a ocorrer tanto na disponibilidade dos recursos quanto em problemas na instalação das máquinas. Esse monitoramento pode também ser utilizado por desenvolvedores de aplicações paralela para identificar as fontes de possíveis gargalos de desempenho nas suas aplicações.

Este capítulo mostra alguns dos problemas relacionados ao monitoramento de *cluster* e à depuração de aplicações paralelas. No final, são descritas algumas ferramentas que se propõem a resolver esses problemas.

2.1 Problemas de Monitoramento de *Cluster*

Nesta seção serão discutidas situações para ilustrar problemas relacionados ao uso de múltiplos monitoradores, dificuldade na correlação dos dados e na existência de domínios administrativos diferentes.

2.1.1 Múltiplos monitoradores

Durante o monitoramento de um *cluster*, é possível que aconteçam situações onde as informações fornecidas por um monitorador não sejam suficientes para que o administrador identifique o problema. Nessas situações, o administrador acaba por utilizar mais de uma ferramenta de monitoramento.

Situações em que o administrador precisa de mais de uma ferramenta são mais complexas porque os monitoradores têm uma independência de funcionamento. Essa independência se caracteriza por vários fatores:

- coleta das informações monitoradas
- registro dos dados durante e após a coleta
- análise ou visualização dos dados

A coleta significa que dois ou mais monitores normalmente têm características independentes na forma como os dados são coletados, já que não há uma forma padronizada para realizar esta tarefa. O registro dos dados normalmente é feito em formatos de arquivos independentes e muitas vezes não documentado. A forma de análise ou visualização normalmente está ligada ao formato dos dados e à forma como foram coletados, não sendo

em geral possível analisar-se conjuntamente informações que tenham sido geradas por outras ferramentas de monitoramento.

A Figura 2.1 mostra uma situação onde o administrador está utilizando três monitoradores independentes para realizar o gerenciamento dos recursos: a ferramenta A é responsável por coletar informações de fluxos de execução de determinados processos; a ferramenta B registra dados sobre a rede de interconexão do *cluster* onde esses processos estão executando enquanto que a ferramenta C obtém dados de cada nó desse *cluster*. Caso o administrador necessite observar o comportamento do *cluster* como um todo, ele vai ter que analisar as informações de cada um dos monitoradores individualmente, mesmo que os dados estejam relacionados entre si. Isso pode prejudicar a identificação da fonte do problema.

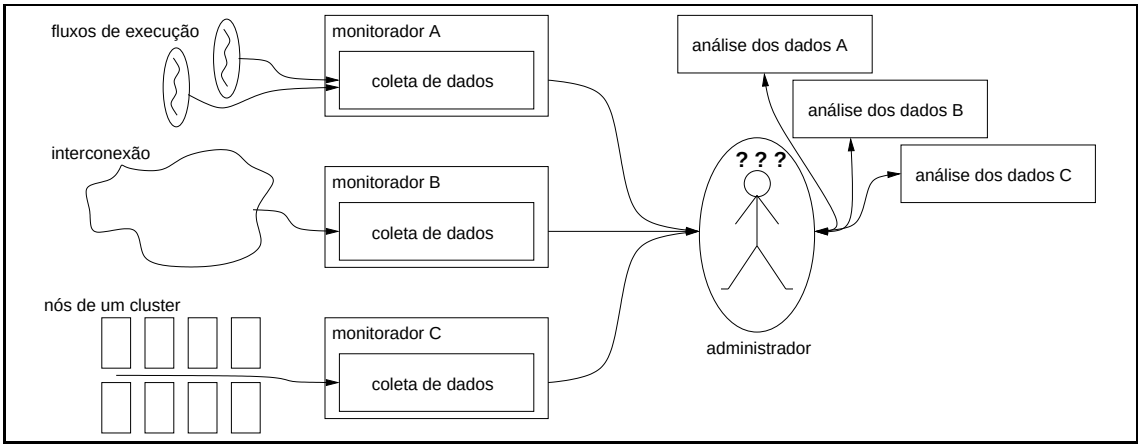

Figura 2.1: Situação onde o administrador deve utilizar mais de uma ferramenta para realizar o monitoramento de um sistema computacional

2.1.2 Dificuldade da correlação de dados

A dificuldade de correlacionar informações se origina na existência de monitoradores de informações independentes, visto na seção anterior. Essa dificuldade ocorre quando as informações que cada um desses monitoradores monitora são relacionadas (STEIGNER; WILKE, 2001; OTTOGALLI et al., 2001). Por exemplo, o programador de uma aplicação paralela está registrando informações durante a execução de sua aplicação. Nesse período de tempo, podem ter acontecido gargalos de desempenho na comunicação entre os processos. A explicação para esse comportamento pode estar no ambiente onde é executada a aplicação, através de outros processos sendo executados concomitantemente.

A correlação das informações oriundas de mais de uma ferramenta de monitoramento implica fundamentalmente na sincronização das informações. Quando se utiliza mais de uma ferramenta, normalmente não se tem um sistema que auxilie na sincronização automática dos eventos e uma posterior correlação entre eles.

2.1.3 Diferentes domínios administrativos

Um *Grid* computacional geralmente é composto por vários domínios administrativos, cada um com regras e possivelmente com ferramentas de monitoramento diferentes. Quando um desenvolvedor de aplicações para esse tipo de sistema deseja monitorar seu programa, ele deve estar preparado para utilizar as diferentes ferramentas disponíveis.

Isso acontece porque em cada um dos domínios provavelmente se utiliza uma ferramenta de monitoramento diferente, com características e funcionamento diferentes, ou não se utiliza.

A Figura 2.2 mostra o desenvolvedor diante de três domínios administrativos diferentes. Ao monitorar a execução de sua aplicação ele provavelmente vai estar diante de mais de uma ferramenta de monitoramento. Ao utilizar outras ferramentas de monitoramento para tentar identificar quedas de desempenho oriundas não na sua aplicação, o desenvolvedor vai ter problemas para correlacionar as informações, pois não há uma integração dos dados.

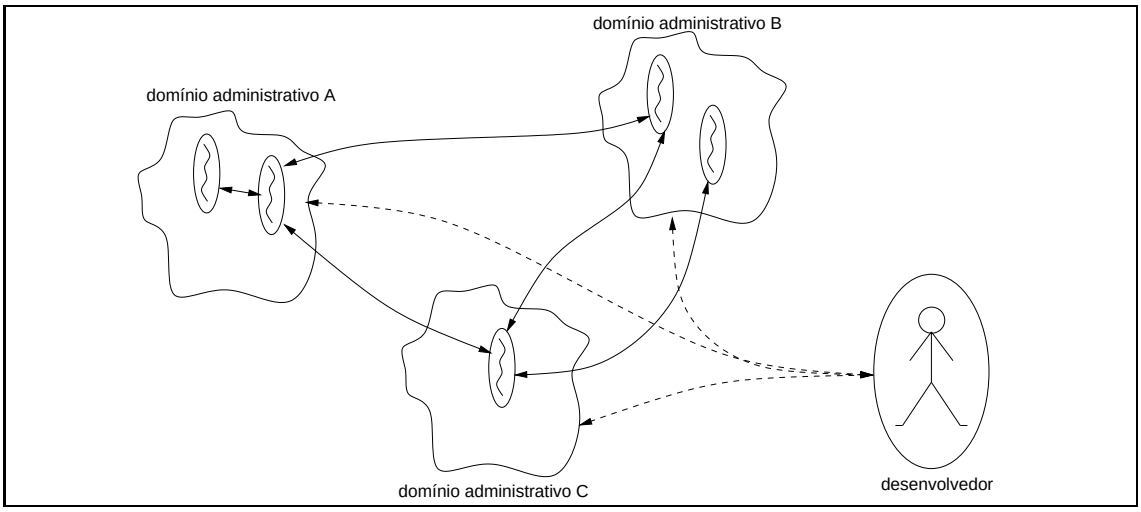

Figura 2.2: Situação onde o desenvolvedor se depara com três domínios administrativos diferentes em um *Grid* com ferramentas de monitoramento diferentes

2.2 Trabalhos Relacionados

Algumas soluções para monitoramento de *cluster* e depuração de aplicações paralelas já existem. Dentre essas soluções, algumas se preocupam em tentar resolver alguns dos problemas mostrados na seção anterior, ou seja, problemas de correlação de informações de diferentes fontes e necessidade de adaptação com mais informações monitoradas. A seguir, serão apresentadas as ferramentas Paraver, Cosmos, Clane e um método para monitoramento de aplicações Java.

2.2.1 Paraver

Paraver (PILLET et al., 1995) é uma ferramenta para depuração de aplicações paralelas. Ela foi desenvolvida na Universidade Técnica da Catalunha, na Espanha, e está disponível apenas comercialmente. O objetivo principal dessa ferramenta é permitir uma visão global do comportamento do programa através da janelas de visualização. A partir dessas visualizações, a ferramenta permite realizar uma análise quantitativa dos dados.

O processo de visualização da ferramenta Paraver é offline, ou seja, o processo de análise de dados de monitoramento é feito em duas etapas: rastreamento e análise. A etapa de rastreamento consiste na integração da aplicação do desenvolvedor com um conjunto de módulos do Paraver. Esses módulos são responsáveis por registrar, durante a execução da aplicação, os principais eventos que acontecem. Atualmente, tem-se módulos para re-

gistro de aplicações desenvolvidas em MPI, em OpenMP, e na combinação dessas duas ferramentas, além de módulos para registro de informações do ambiente, como SCPU, que captura as rotinas de escalonamento de processos, e InfoPerfex, utilizado para capturar e registrar os contadores de hardware de arquiteturas SGI. A segunda etapa, de análise, acontece após o fim da primeira etapa e consiste inicialmente na leitura dos arquivos registrados previamente pela ferramenta Paraver.

A ferramenta Paraver recebe como entrada o conjunto de arquivos registrado na primeira etapa de rastreamento e arquivos de configuração. A partir de então, a ferramenta lê os dados desses arquivos e passa por um conjunto de módulos internos à ferramenta. A Figura 2.3 ilustra a organização desses módulos. O módulo de filtro tem acesso direto ao arquivo com os rastros e seu objetivo é fornecer uma visão parcial configurável das informações registradas. A configuração desse módulo se dá através de um dos arquivos de configuração fornecidos pelo desenvolvedor. O módulo semântico calcula os valores que serão transferidos aos módulos de representação dos dados. O módulo de semântica é o único que está ligado diretamente a origem dos dados, ou seja, se a origem dos dados é, por exemplo, a semântica MPI, deve haver um módulo de semântica apropriado para eventos oriundos do módulo de monitoramento MPI. O desenvolvedor pode configurar e personalizar este módulo para obter uma visualização particular desejada. O último módulo, de representação de dados, é composto por três sub-módulos: visualização, textual e de análise. O sub-módulo de visualização mantém um diagrama temporal com as informações de estado e eventos dispostos graficamente. Os fluxos de execução são mostrados por esse sub-módulo através de barras horizontais na janela de visualização. O sub-módulo de análise mostra dados estatísticos através de tabelas e histogramas e pode ser configurado para se analisar apenas uma faixa de tempo. O último sub-módulo, textual, mostra informações específicas relacionadas ao rastro de execução de uma aplicação paralela.

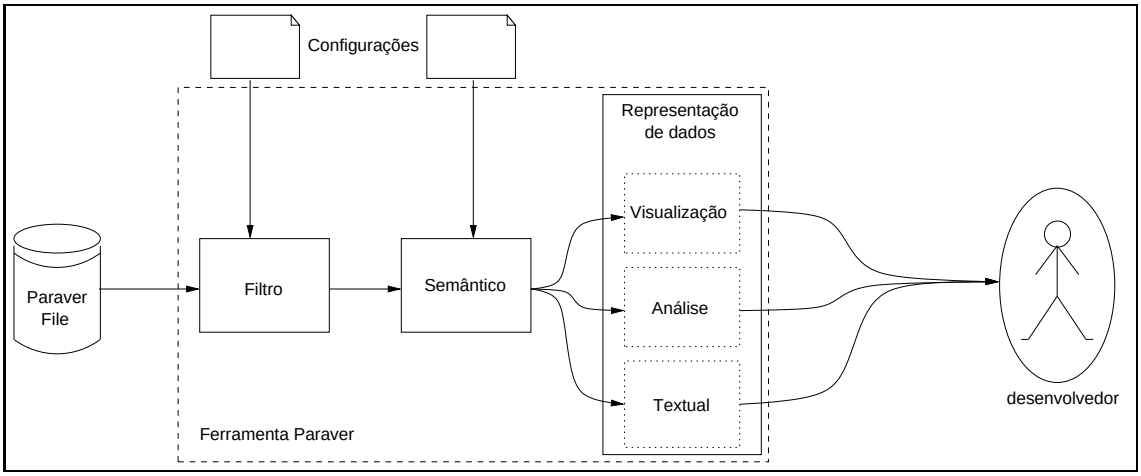

Figura 2.3: Funcionamento da ferramenta Paraver desde o recebimento de arquivos de rastreamento até a visualização e análise dos dados pelo desenvolvedor

A combinação da utilização de diferentes módulos de rastreamento com a configurabilidade da ferramenta Paraver permite a análise do comportamento de uma aplicação paralela utilizando não só dados da aplicação em si, mas também do ambiente. Esses dados são correlacionados automaticamente pela ferramenta através da sincronização das informações durante o processo de análise. Esse funcionamento resolve os problemas de

múltiplos monitoradores, na seção 2.1.1, e da dificuldade de correlacionamento das informações, na seção 2.1.2, mas somente para a depuração de aplicações paralelas. Um problema da ferramenta é que ela somente trabalha com um formato específico de arquivo. Caso o usuário necessite outras informações disponíveis em outro formato, ele deve primeiro convertê-las para o formato da ferramenta Paraver.

2.2.2 *CoSMoS*

CoSMoS - Coblenz Software Monitoring System (STEIGNER; WILKE, 2001) é um sistema integrado de monitoramento de aplicações paralelas. Foi desenvolvido na Universidade de Koblenz-Landau, na Alemanha. A principal justificativa da criação do *CoSMoS* é que o desempenho de aplicações paralelas são influenciados também por fatores externos à aplicação, como a disponibilidade de recursos e o escalonamento do sistema operacional. Dessa forma, o *CoSMoS* pode correlacionar dados oriundos da aplicação e de fontes externas de informação, permitindo um entendimento mais completo do comportamento da aplicação durante a execução.

A arquitetura do *CoSMoS* é, assim como a ferramenta Paraver da seção anterior, dividida em duas partes: rastreamento de informações e análise/visualização dos dados. Essa arquitetura está ilustrada na Figura 2.4. A etapa de rastreamento é feita por três componentes: o monitor da aplicação, o monitor de rede/máquina e o monitor do sistema operacional. As informações rastreadas por esses três componentes são gravadas em arquivo durante o processo de rastreamento. Após o fim do período de monitoramento, os arquivos gerados podem ser utilizados pelo analisador, que faz a correlação das informações registradas e grava os dados integrados em uma base de dados SQL. Os dados registrados no período de monitoramento são visualizados então por uma ferramenta de visualização.

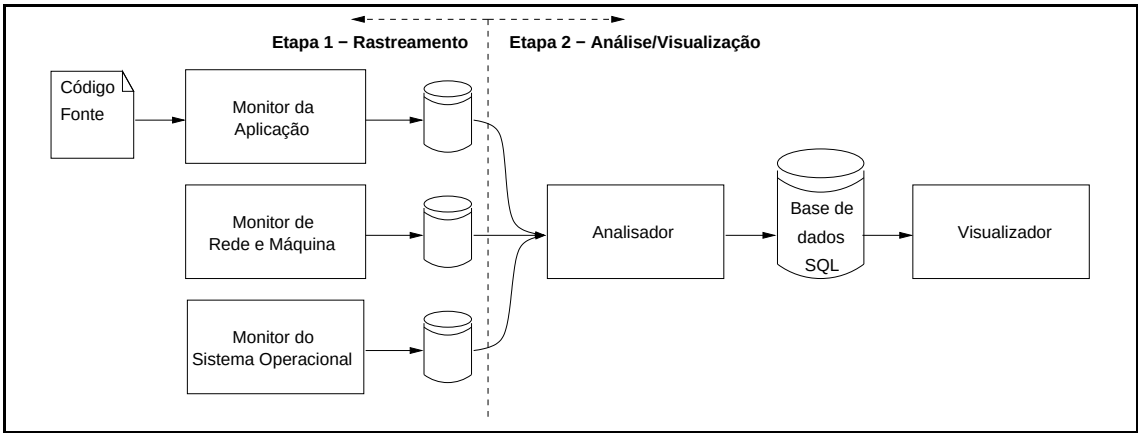

Figura 2.4: Arquitetura do *CoSMoS* desde o rastreamento das informações dos três níveis (aplicação, rede/máquina e sistema operacional) até a visualização

A primeira etapa do processo de monitoramento do *CoSMoS* consiste em obter informações através de três componentes diferentes: monitor da aplicação, de rede/máquina e do sistema operacional. O monitor da aplicação funciona através da técnica de instrumentação. Para realizar essa instrumentação, a ferramenta *CoSMoS* fornece um conjunto de funções e uma aplicação gráfica que facilita esse processo. Após o período de instrumentação, a aplicação do programador é ligada a uma biblioteca, de acordo com a linguagem e o paradigma de programação utilizados. Atualmente, o *CoSMoS* tem suporte

às linguagens C, C++ e Java. A função dessa biblioteca é, durante a execução, receber os registros feitos através das funções de instrumentação e enviá-las a um servidor. Esse servidor recebe informações de todas as instâncias de uma aplicação paralela e as registra em arquivo.

Outro componente da primeira etapa do processo de monitoramento do *CoSMoS* é o monitor de rede/máquina. Esse monitor tem por objetivo rastrear informações sobre a alocação de recursos das máquinas que fazem parte do ambiente onde a aplicação a ser monitorada é executada. Para funcionar, esse monitor tem um agente em cada uma das máquinas que fica periodicamente monitorando recursos como, por exemplo, a utilização da CPU, de memória e de rede. Os agentes se comunicam entre si periodicamente possibilitando a construção de uma estrutura hierárquica, onde cada nível é gerenciado por um agente controlador. Essa estrutura hierárquica possibilita uma maior simplicidade ao configurar todos os agentes antes de se iniciar um período de monitoramento. Após o início do período de monitoramento, os agente registram os dados monitorados em um arquivo local à máquina onde está executando.

O último componente da etapa de rastreamento é o monitor do sistema operacional. O *CoSMoS* oferece uma interface para uma outra ferramenta, chamada FKT - *Fast Kernel Tracing* (RUSSELL; CHAVAN, 2002), que tem por objetivo realizar o registro das mudanças de estado de um conjunto de processos dentro do sistema operacional. As informações dos monitores de sistema operacional de cada máquina do ambiente de execução são registradas em memória dentro do próprio sistema operacional. Após o fim do período de monitoramento, os dados são registrados em arquivo.

A utilização desses três monitores, ou seja, do monitor de aplicação, de rede/máquina e do sistema operacional oferece ao desenvolvedor de aplicações paralela informações oriundas de três níveis de abstração diferentes do sistema de execução. A justificativa do *CoSMoS* para se ter esse três tipos de monitoradores é que a explicação para o comportamento das aplicações paralelas pode estar em lugares diferentes, nem sempre somente em um desses níveis. Para que a informação coletada pelos três componentes de rastreamento seja útil para o desenvolvedor, tarefas como a sincronização e a integração dos dados devem ser realizadas antes da visualização. Essas tarefas são feitas na segunda etapa do processo de monitoramento do *CoSMoS*, que é a da análise/visualização.

A segunda etapa do processo de monitoramento do *CoSMoS* consiste em sincronizar as informações coletadas pelas ferramentas de rastreamento e permitir que a visualização dessas informações sejam feitas conjuntamente. A integração dos dados é feita pelo componente analisador, que pode ser observado na Figura 2.4. Esse componente lê as informações registradas pelos três monitoradores, realiza a sincronização e integração dos dados e as grava em uma base de dados SQL. O último componente do *CoSMoS* é o visualizador gráfico. Esse visualizador é composto por gráficos estáticos e dinâmicos. O estático é responsável por apresentar dados estatísticos acerca das informações coletadas enquanto que o dinâmico mostra as informações dos três monitores de forma integrada em uma janela de visualização.

Assim como a ferramenta Paraver, o *CoSMoS* permite o monitoramento da aplicação e a análise dos dados de forma conjunta, podendo ser feita a correlação entre as informações coletadas por diferentes módulos. A diferença entre as duas ferramentas é que o *CoSMoS* oferece uma estrutura completa de monitoramento, enquanto que o Paraver permite a utilização de um conjunto de módulos para o rastreamento, sendo então mais flexível principalmente no que tange a extensibilidade.

Em relação aos problemas apresentados na seção anterior, o *CoSMoS* resolve os pro-

blemas de correlação das informações coletadas por diferentes fontes. Entretanto, ele não oferece uma forma de estender as fontes de informação.

2.2.3 *Clane*

Clane (FERRETO; ROSE, 2003) é um ambiente de análise de desempenho para *clusters*. Foi criado em 2003 no Centro de Pesquisa em Alto Desempenho da Universidade Católica do Rio Grande do Sul. O objetivo da ferramenta *Clane* é combinar informações de gerenciamento e monitoramento de recursos de um *cluster* relacionadas às aplicações que são executadas nesse *cluster*. Acredita-se que essa integração permita uma melhor análise da aplicação e do sistema (FERRETO; ROSE, 2003).

A arquitetura da ferramenta *Clane*, ilustrada na Figura 2.5, é composta basicamente por dois componentes: o servidor de informação e a ferramenta de análise. O servidor de informação é responsável por realizar a coleta de informações tanto do gerenciador quanto do monitor do *cluster*. Essas informações são obtidas através da utilização, por parte do gerenciador e monitor, de funções específicas do servidor de informação. Essas funções são disponibilizadas através de uma biblioteca que pode ser ligada às ferramentas de gerenciamento de monitoramento. Isso implica que essas ferramentas devem ser alteradas para que sejam integradas ao servidor de informação do *Clane*.

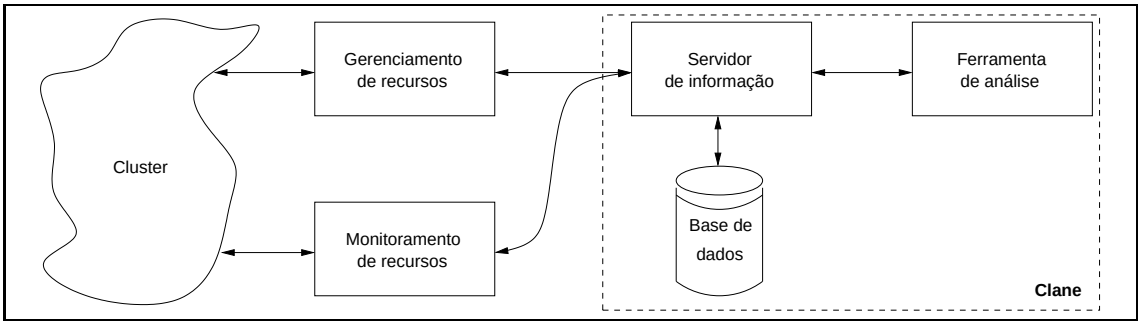

Figura 2.5: Arquitetura do *Clane* com a conexão com o gerenciador e o monitor de um *cluster*

O outro componente da arquitetura do *Clane* é a ferramenta de análise. Essa ferramenta utiliza uma biblioteca que possibilita a coleta de dados do servidor de informação. As informações obtidas podem ser filtradas com o conjunto de funções fornecido pela biblioteca e então visualizadas graficamente e estatisticamente pela ferramenta de visualização.

O funcionamento desse sistema se dá principalmente através da integração das ferramentas de monitoramento e gerenciamento do *cluster* com o *Clane*. Essa integração permite que essas ferramentas periodicamente enviem dados para o servidor de informação. Esse servidor de informação registra os dados recebidos em uma base de dados com formato XML. Para realizar a visualização dos dados registrados, o desenvolvedor ou usuário do *cluster* deve utilizar o componente de análise do *Clane*. Esse componente se conecta ao servidor de informação e obtém os dados previamente registrados.

A ferramenta *Clane* foi integrada com o monitorador RVision (FERRETO; ROSE; ROSE, 2002) e o gerenciador de *cluster* Crono (NETTO; ROSE, 2003). Essa integração foi feita instrumentando-se as ferramentas com as funções disponibilizadas pela biblioteca que faz a interface com o servidor de informações.

Uma característica positiva da ferramenta *Clane* é que ela é capaz de integrar informações de duas fontes diferentes de informação. Essa integração, no entanto, parece ser limitada por não haver uma forma clara de como fazer para estender a ferramenta caso se utilize mais de um monitorador no *cluster*, o que eventualmente pode ser útil quando são necessárias informações complementares de diferentes ferramentas de monitoramento. O *Clane* é a primeira ferramenta vista nesta seção que pode ser utilizada tanto na área de depuração de aplicações paralelas quanto na área de administração de *cluster*. As ferramentas *CoSMoS* e Paraver, ao contrário, se focam apenas na depuração de aplicações paralelas.

2.2.4 Rastreamento Java Multi-nível

Utilizar invocação remota de métodos (RMI) do Java é uma forma de desenvolver aplicações distribuídas. O monitoramento dessas aplicações pode ser feito através da interface JVMPI. No entanto, esta interface não permite a combinação dos pontos de comunicação do RMI quando são utilizadas duas máquinas virtuais Java. O trabalho descrito nesta seção (OTTOGALLI et al., 2001) apresenta uma solução para esse problema e permite a visualização dos dados rastreados em Pajé (KERGOMMEAUX; OLIVEIRA STEIN, 2003).

O problema de rastreamento de aplicações Java com comunicação entre as máquinas virtuais é feito através da coleta de informações em diferentes níveis de abstração do sistema computacional, como por exemplo o nível da aplicação e do sistema operacional. Através dessa coleta multi-nível, é possível resolver o problema de identificação das comunicações entre duas máquinas virtuais Java. Assim, as informações são gravadas pela JVMPI no nível de abstração da aplicação e pelo sistema operacional no nível mais baixo de abstração. Esse nível mais baixo de abstração se dá através da utilização de um *middleware* chamado *Jonathan* (DUMANT et al., 1998). As comunicações RMI são implementadas neste middleware de forma que ele pode ser alterado para registrar informações dos pontos de comunicação utilizados pelo aplicação. Além de dados para identificar as comunicações entre dois pontos de comunicação RMI, a coleta de dados no nível mais baixo permite obter informações sobre a utilização de recursos gerenciados pelo sistema operacional.

A Figura 2.6 mostra a arquitetura da ferramenta. O rastreamento ocorre durante a execução da aplicação Java distribuída. Cada fluxo de execução tem os principais eventos capturados pela JVMPI e então registrados em arquivo. Em um segundo momento há o processo de integração dos dados. Nessa integração, as informações oriundas no *middleware Jonathan* são integradas com os dados capturados pela interface JVMPI. Durante esse processo de integração, os pontos de uma comunicação RMI são combinados e então é possível realizar a visualização dessas comunicações. O resultado do processo de integração é um arquivo que pode ser visualizado na ferramenta Pajé.

O foco desse processo de integração de informações é o rastreamento de aplicações Java Multi-nível. A integração de dados realiza a sincronização dos eventos e a correlação de informações entre o nível da aplicação Java e o nível mais próximo ao sistema operacional, exercido pelo *Middleware Jonathan*. No entanto, esse processo de integração descrito por OTTOGALLI et al. (2001) é específico para aplicações Java que utilizam o *Middleware Jonathan*. Além disso, o processo não descreve uma forma de se adicionar novas fontes de informação na integração de dados. Atualmente, o JVMPI foi substituído pela interface JVMTI, de forma que esse rastreamento multi-nível só pode ser utilizado com máquinas virtuais Java antigas.

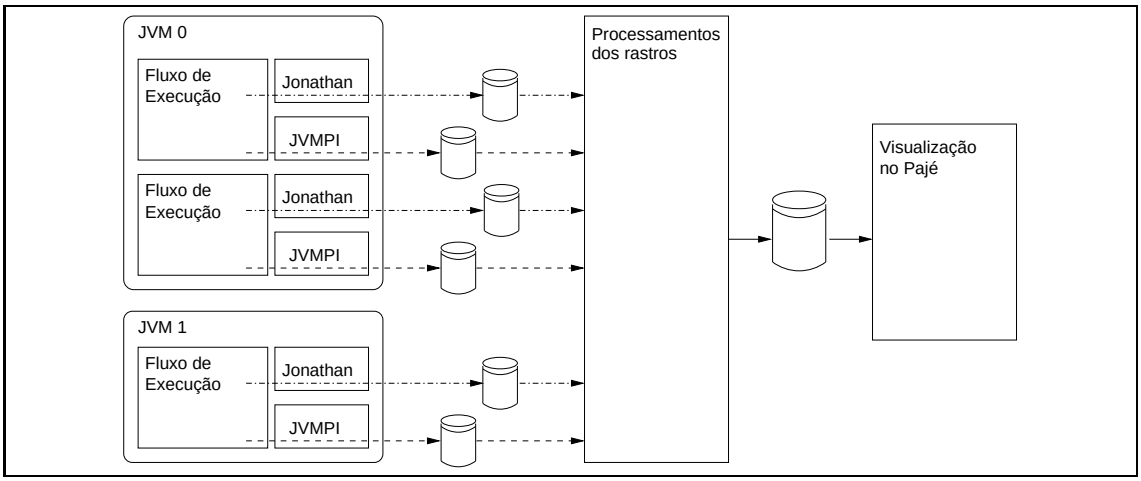

Figura 2.6: Arquitetura do processo de rastreamento multi-nível de aplicações Java com comunicações RMI entre duas ou mais máquinas virtuais Java

Por outro lado, o rastreamento de aplicações Java com dados de diferentes níveis do sistema computacional apresenta a necessidade de se coletar dados por diferentes fontes de informação para se obter uma melhor visão do comportamento de um programa Java. Além disso, o trabalho utiliza como ferramenta de visualização o Pajé, mostrando a adaptação e extensibilidade a diferentes semânticas de programação dessa ferramenta de visualização. Existe outro trabalho semelhante ao desenvolvido por OTTOGALLI et al. (2001) que também utiliza Pajé (SILVA; SCHNORR; STEIN, 2003).

2.2.5 Outras ferramentas

Grande parte das soluções da área de monitoramento de *cluster* como Ganglia (SACERDOTI; MASSIE; CULLER, 2003), Performance Co-Pilot (PCP) (SGI, 1999), ClusterProbe (LIANG; SUN; WANG, 1999), Supermon (SOTTILE; MINNICH, 2002), Parmon (BUYYA, 2000) e RVision (FERRETO; ROSE; ROSE, 2002) funcionam de forma independente. Essas ferramentas em geral não possibilitam uma integração com informações coletadas por outras ferramentas de monitoramento de *cluster*.

Na área de depuração da aplicações paralelas, a maioria das ferramentas está atrelada a uma determinada semântica de informações em um determinado formato. Dentre essas ferramentas, pode-se citar Pablo (ROSE; ZHANG; REED, 1998), Paradyn (MILLER et al., 1995) e Vampir (NAGEL et al., 1996; BRUNST et al., 2001). Em geral, essas ferramentas de depuração de aplicações não oferecem a possibilidade de integrar os dados coletados por elas com dados coletados por outras ferramentas.

Alguns trabalhos (NEMETH; GOMBAS; BALATON, 2004; TIERNEY et al., 2002) discutem a utilização de um sistema de monitoramento para *Grids* onde são utilizados coletores em diferentes níveis do sistema computacional. Além desses trabalhos, existe uma solução extensível para o monitoramento de recursos de um *Grid* (BAKER; SMITH, 2002). Entretanto, esta solução não aborda a correlação desses dados coletados com informações das aplicações que podem ser executadas neste *Grid*.

2.3 Conclusão do capítulo

Dentre as soluções para monitoramento de *cluster* e depuração de aplicações paralelas vistas neste capítulo, a maioria não permite uma fácil extensibilidade ou adaptação a diferentes sistemas de monitoramento. A ferramenta *CoSMoS*, por exemplo, tem dados oriundos de diferentes fontes de informação mas não permite a adição de novas ferramentas de coleta de dados. A ferramenta Paraver, por outro lado, permite a adição de novos módulos mas seu foco é a depuração paralela, excluindo o monitoramento de *cluster*. Além disso, não foi encontrada nenhuma ferramenta que permita adaptação de dados de monitoramento coletados por programas já existentes. No próximo capítulo será apresentado o modelo integrado de monitoramente de *cluster* proposto neste trabalho.

3 PROPOSTA DE MODELO INTEGRADO DE MONITORAMENTO

No monitoramento de um *cluster*, o administrador pode estar diante de uma situação onde mais de uma ferramenta de monitoramento é necessária para que ele possa tomar uma decisão acerca de problemas que podem estar acontecendo. Nestas situações, é interessante obter uma ferramenta que realize a integração dos dados das diferentes ferramentas de monitoramento e permita uma visualização integrada desses dados. Essa visualização pode permitir ao administrador verificar a correlação entre os dados complementares das diferentes ferramentas de monitoramento e encontrar mais facilmente as fontes de problemas.

Na área de depuração de aplicações paralelas, é importante aliar dados da execução de aplicações com informações do ambiente onde são executadas as aplicações paralelas (KARAVANIC et al., 1997; OTTOGALLI et al., 2001; AGARWALA et al., 2003). Para tornar úteis as informações coletadas no ambiente, é necessário sincronizar e correlacionar os dados de forma que o desenvolvedor perceba melhor as causas de determinados comportamentos em suas aplicações paralelas. É importante que a ferramenta que faça essa sincronização seja extensível, de forma que se adapte às necessidades do desenvolvedor e permita a utilização de dados de outras ferramentas de monitoramento.

A utilização de uma ferramenta que permita a utilização de múltiplas fontes de informação é particularmente útil quando se está realizando o rastreamento de aplicações executadas em um *Grid*. Nestes ambientes, é normal a existência de diferentes domínios administrativos com ferramentas de monitoramento diferentes onde a troca dessas ferramentas não é uma opção.

Este capítulo descreve a proposta de um modelo integrado de monitoramento que pode ser utilizado tanto para o **monitoramento de *cluster*** quanto para a **depuração de aplicações paralelas**. Nas seções seguintes serão apresentados o modelo e as etapas de coleta e de integração dos dados desse modelo.

3.1 Modelo de Integração de Dados

O objetivo do modelo é permitir a integração de dados coletados por diferentes fontes de informação e, por fim, permitir a visualização conjunta desses dados. Como fontes de informação, o modelo propõe a utilização de ferramentas de monitoramento e bibliotecas de rastreamento já existentes. Dessa forma, o modelo deve poder se adaptar a diferentes situações onde são necessárias múltiplas fontes de dados. Exemplos dessas diferentes situações são o monitoramento de *cluster* e a depuração de aplicações paralelas. O modelo

é baseado na depuração *offline* de aplicações paralelas, onde o processo de visualização dos dados ocorre em duas etapas: a etapa de **coleta** e a etapa de **integração** dos dados coletados.

A divisão do processo de monitoramento em duas etapas é utilizado por outras ferramentas de monitoramento, como *CoSMoS* (STEIGNER; WILKE, 2001) e Paraver (PILLET et al., 1995). A vantagem de se utilizar esse processo em duas fases é que, no monitoramento de aplicações paralelas, a intrusão que acontece por causa do monitoramento pode ser minimizada. No caso do monitoramento de um *cluster*, a vantagem é que é possível comparar períodos de monitoramento diferentes ao longo do tempo, como é feito na ferramenta *Clane* (FERRETO; ROSE, 2003).

A Figura 3.1 mostra a visão geral do modelo. A etapa 1, de coleta, é realizada durante o período de tempo que se deseja monitorar o sistema computacional. Ela consiste na coleta de informações por diferentes ferramentas de monitoramento ou de bibliotecas de rastreamento de aplicações. O resultado dessa etapa é um conjunto de arquivos com os dados que foram registrados. A etapa 2, de integração, consiste basicamente na união das informações registradas de forma que possam ser visualizadas e analisadas conjuntamente. Essa etapa acontece depois do fim do período de monitoramento.

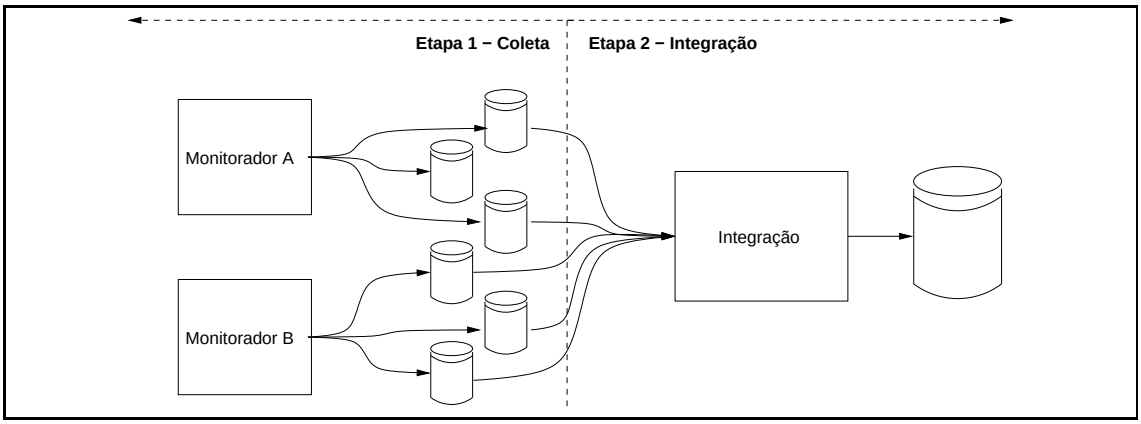

Figura 3.1: Visão geral do modelo com as duas etapas: coleta e integração.

3.2 Etapa de Coleta

A etapa de coleta consiste no registro de um conjunto de informações durante o período de monitoramento. Essas informações podem ser monitoradas por um número variável de ferramentas de monitoramento de *cluster* ou bibliotecas de rastreamento de informações já existentes. Por exemplo, pode-se utilizar ferramentas de monitoramento como o Ganglia (MASSIE; CHUN; CULLER, 2004) e RMS (UTHAYOPAS; PHATANAPHEROM, 2001) com o rastreamento de aplicações através de bibliotecas de rastreamento como libRastro (SILVA; OLIVEIRA STEIN, 2002). Essas ferramentas monitoram e rastreiam durante o período de monitoramento que se deseja observar. O resultado desta etapa deve ser um conjunto de arquivos com as informações que foram registradas pelas diversas ferramentas utilizadas.

A restrição caracterizada neste modelo acerca do formato dos arquivos de rastreamento é que se tenha os registros separados em eventos, sendo que cada evento deve ter obrigatoriamente um tipo que identifica esse evento e a data em que aconteceu. Além dessas informações obrigatórias, os eventos podem possuir outras informações adicionais, de

acordo com o tipo do evento. Embora o modelo imponha esta restrição, a maioria das ferramentas de monitoramento e bibliotecas de rastreamento utiliza eventos tipados em seus formatos de arquivo.

3.3 Etapa de Integração

Esta seção apresenta a segunda parte do modelo, ou seja, o processo de integração dos dados. Essa integração utiliza as informações registradas no período de monitoramento pelas ferramentas de coleta, além das definições de hierarquia de entidades, forma dos identificadores e dados de sincronização.

A Figura 3.2 mostra a modelagem dessa segunda parte do modelo. Os dados gerados pela etapa de coleta são fornecidos ao modelo de integração de dados. A primeira sub-etapa deste modelo é a sincronização das informações, responsável por sincronizar as datas dos eventos das múltiplas fontes de informação de acordo com um relógio de referência único. Depois são realizadas a unificação de identificadores e da hierarquia das entidades, que trata da união semântica das informações. Por fim, é realizada padronização de formato, que consiste basicamente no mapeamento dos eventos de cada fonte de dados em eventos de uma determinada ferramenta de visualização.

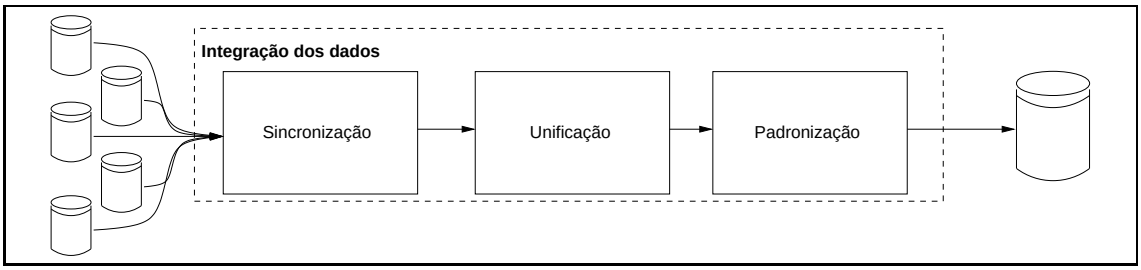

Figura 3.2: Modelo do processo de integração

As próximas seções apresentam os detalhes do modelo para cada sub-parte da etapa de integração de informações, que são a sincronização, unificação e padronização das informações.

3.3.1 Sincronização

A sincronização dos eventos gerados pelas diferentes fontes de dados é importante para manter a relação causal e/ou temporal entre esses eventos. Essa sincronização deve acontecer porque em sistemas como *clusters* normalmente não há um relógio global.

Na etapa de coleta do modelo, apresentada na seção anterior, cada ferramenta de monitoramento utiliza um determinado relógio para registrar seus eventos. A data desses eventos são usualmente relativos à data de um determinado evento. Por exemplo, algumas ferramentas registram a data dos eventos relacionada à data do primeiro evento gerado pela ferramenta durante o período de monitoramento. Outras utilizam a data relativa a um evento anterior ao período de monitoramento. Existem também algumas que registram a data do relógio local ao computador onde o evento ocorreu.

Para realizar a sincronização, deve-se portanto identificar qual foi o relógio de referência utilizado para datar esse evento. Com essas informações de cada ferramenta de monitoramento utilizada no período de coleta, deve-se escolher um determinado relógio de referência para realizar a conversão das datas de todos os eventos com suas datas

relativas para a data desse relógio de referência. Caso uma determinada ferramenta de monitoramento utilize mais de uma data relativa para registrar os eventos, deve-se realizar a identificação dessas datas relativas e converter a datas dos eventos registrados com cada uma delas para a data do relógio de referência escolhido.

Existem várias técnicas para a sincronização de relógios (LAMPORT, 1978; KOPETZ; OCHSENREITER, 1987). A maioria dessas técnicas exige a alteração das ferramentas de monitoramento para que possam ser registrados datas sincronizadas entre os diversos eventos. Além da complexidade de se alterar todas as ferramentas que possam vir a ser utilizadas em uma determinada ocasião, o modelo de integração de dados tem como um dos objetivos não alterar as ferramentas de coleta de dados que utiliza como fonte de informação. Além disso, cada uma das ferramentas de coleta normalmente utiliza um método de sincronização interno e diferentes das outras ferramentas. Devido a essas características, deve haver um método externo e independente para a correção das datas dos eventos registrados por diferentes ferramentas de coleta. Uma forma de realizar a sincronização dos eventos de forma independente e externa às aplicações é utilizar o método descrito no artigo de MAILLET; TRON (1995). Esse método consiste em inicialmente escolher uma determinada máquina do sistema computacional. Essa máquina pode ou não fazer parte do sistema que será monitorado e seu relógio será usado como referência para a sincronização dos eventos. Antes do período de monitoramento, são registradas as diferenças de relógio entre essa máquina de referência e todas as outras máquinas que devem ser monitoradas. Essa tarefa deve ser feita na etapa de coleta do modelo. Ao fim deste processo, as diferenças entre os relógios da máquina de referência e das máquinas monitoradas são novamente obtidos. O resultado final desse processo de obtenção das diferenças de relógio é um arquivo que é utilizado para se sincronizar os eventos registrados pelas diferentes ferramentas de monitoramento utilizadas na etapa de coleta. Com esse arquivo, cada evento, registrado com o relógio de uma determinada máquina, é sincronizado de acordo com a diferença de relógio dessa máquina com o relógio da máquina de referência. O registro das diferenças de relógio ao fim do período de monitoramento é feito para levar em conta na sincronização a defasagem (*drift*) dos relógios. Como são feitas apenas duas medidas de diferenças de relógio, uma no início e outra no fim do período de monitoramento, a correção dessa defasagem é linear. Dessa forma, o método tem melhores resultados quando são utilizados períodos de monitoramento curtos.

Algumas ferramentas de monitoramento podem já fazer a sincronização dos seus eventos de acordo com o relógio de uma das máquinas. Neste caso, basta refazer a sincronização desses eventos com a máquina de referência escolhida no método descrito acima.

3.3.2 Unificação

A segunda sub-etapa da integração das informações é a unificação das hierarquias de entidades de cada ferramenta de monitoramento ou biblioteca de rastreamento. Uma entidade pode ser qualquer objeto que está sendo monitorado, tanto diretamente quanto indiretamente, ou qualquer valor ou estado que um determinado objeto pode ter em um instante ou período de tempo. Um processo, a utilização de CPU, uma máquina, o estado de bloqueado de um fluxo de execução são exemplos de entidades. A hierarquia de entidades se refere a como as entidades se relacionam. O lado esquerdo da Figura 3.3 ilustra um exemplo de hierarquia de entidades no qual uma máquina pode conter vários processos e cada processo pode estar em dois estados diferentes, bloqueado ou executando. Esse exemplo de hierarquia de entidades é bastante utilizado por bibliotecas de rastreamento de processos. No modelo do sistema que está sendo proposto deve haver

uma forma de definir a hierarquia de entidades para cada ferramenta de monitoramento utilizada na etapa da coleta.

Cada ferramenta de monitoramento normalmente utiliza um conjunto de identificadores para cada entidade da hierarquia de entidades. Esses identificadores têm por objetivo individualizar cada objeto monitorado da hierarquia. O lado direito da Figura 3.3 mostra alguns exemplos de identificadores de máquinas e de processos que podem ser utilizados por uma ferramenta de monitoramento. O modelo de coleta utilizado neste trabalho não especifica um conjunto de identificadores que deve ser utilizado, desde que haja uma forma de descrevê-los de forma que possam ser convertidos, caso necessário, na etapa de integração.

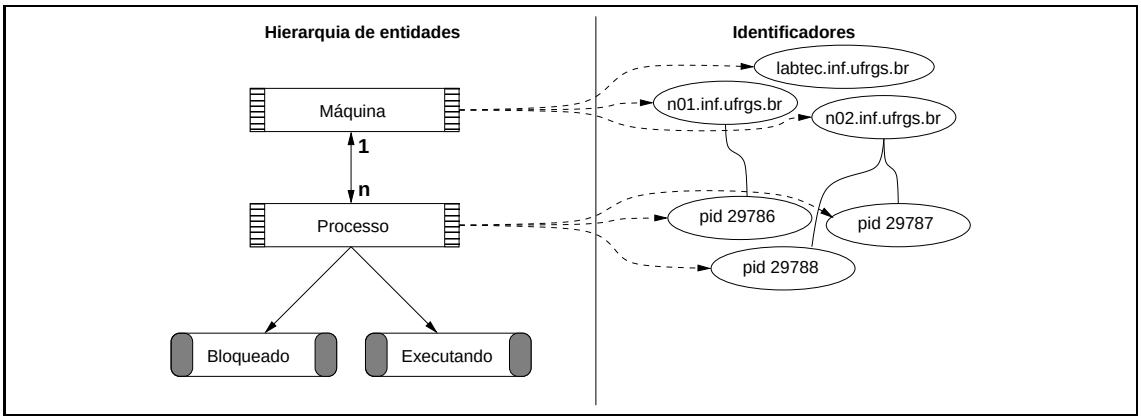

Figura 3.3: Exemplo de hierarquia de entidades e alguns identificadores na modelagem dos rastreadores do sistema

A unificação das informações consiste na integração das hierarquia de entidades e da união dos identificadores de cada uma das ferramentas de coleta utilizadas. Deve-se fazer com que os identificadores de um determinado tipo de entidade sejam os mesmos utilizados por ferramentas de monitoramento diferentes. A tarefa de unificação hierárquica e de identificadores é essencial no processo de integração pois é através dessa unificação que é possível realizar uma visualização conjunta dos dados.

A Figura 3.4 mostra um exemplo de unificação de hierarquia de entidades de duas ferramentas de monitoramento diferentes. A ferramenta A, mostrada no lado esquerdo, tem como hierarquia para cada *cluster* um conjunto de máquinas e para cada máquina um conjunto de informações como dados de CPU, memória, etc. A ferramenta B, do lado direito, tem para cada *cluster* um conjunto de máquinas, sendo que cada máquina tem um conjunto de processos e cada processo pode estar ou no estado executando ou no estado bloqueado. As entidades semelhantes que se pode detectar neste exemplo são a entidade *cluster* e a entidade máquina. A Figura 3.5 mostra as hierarquias de entidades da Figura 3.4 unidas de acordo com as entidades semelhantes detectadas.

O próximo passo na unificação é utilizar os mesmos identificadores para as entidades que foram unidas no processo de unificação de hierarquias. No caso do exemplo anterior da Figura 3.4, as entidades *cluster* e máquina foram unificadas. Se por exemplo a ferramenta A utilizar como identificador de máquina o número IP e a ferramenta B utilizar o nome da máquina, deve haver uma forma de unificar esses identificadores. Essa unificação pode ser feita através de arquivos de configuração ou através de um pré-processamento do registro de uma das ferramentas de monitoramento. Esse mesmo processo de unificação de identificadores deve acontecer com a entidade *cluster* no exemplo utilizado.

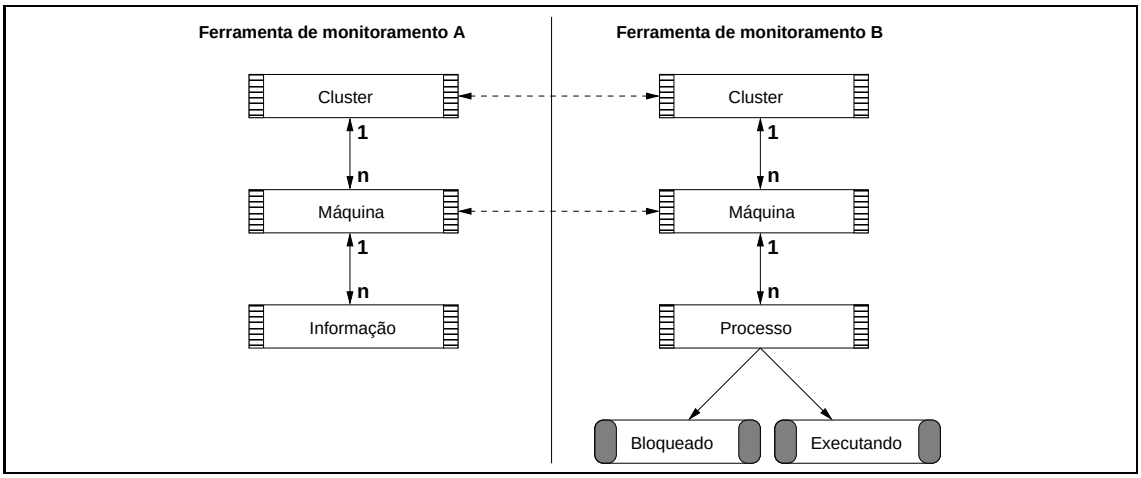

Figura 3.4: Hierarquia de entidades de duas ferramentas de monitoramento

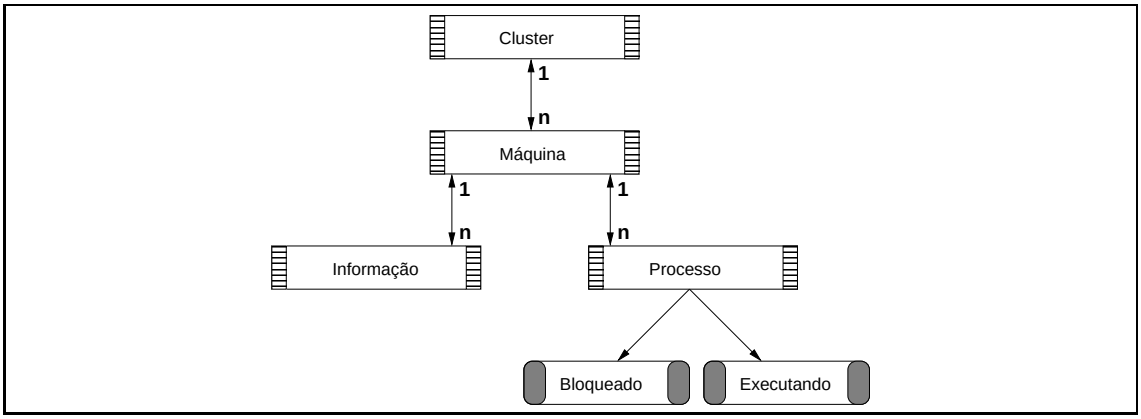

Figura 3.5: Unificação das hierarquias de entidades da Figura 3.4

Em algumas situações a hierarquia de entidades pode ser semelhante em duas ferramentas utilizadas no período de coleta mas as entidades dessas hierarquias podem ter significados diferentes. Por exemplo, dois tipos de ferramentas podem ter a entidade "máquina" em suas hierarquias mas em apenas uma delas essa entidade representa a máquina fisicamente. Na outra, por exemplo, pode-se ter várias instâncias da entidade "máquina" em uma máquina física. Nesses casos, a unificação de identificadores deve realizar um tratamento de forma a construir uma hierarquia que represente a unificação de entidades semelhantes mas com significados diferentes. Esse tratamento consiste basicamente em escolher uma das entidades e converter a outra com significado diferente para essa entidade, através da unificação dos identificadores.

3.3.3 Padronização

A padronização das informações é a terceira sub-etapa da integração dos dados. Essa padronização consiste em transformar a forma de registro dos dados de cada ferramenta de coleta utilizada na primeira etapa do modelo para um formato padrão passível de visualização. O processo de padronização deve utilizar os dados sincronizados, os identificadores e a hierarquia de entidades unificada.

No processo de padronização dos dados de cada ferramenta, deve-se primeiramente

definir qual é a ferramenta de visualização que será utilizada para visualizar os dados. Com base nisso, deve-se definir qual a visualização desejada dos dados das ferramentas de coleta utilizadas de acordo com as informações que foram registradas durante o processo de coleta. Por exemplo, dependendo da ferramenta de visualização escolhida, pode-se mostrar os dados em um gráfico temporal onde diferentes estados das entidades são representados por cores diferentes e dados quantitativos são mostrados por barras horizontais.

Após a definição da ferramenta de visualização que será utilizada e de como se deseja visualizar os dados, deve-se realizar um mapeamento dos eventos gerados pelas diferentes ferramentas para eventos da ferramenta de visualização. Esse mapeamento normalmente é feito especificamente para uma determinada ferramenta de visualização, pois a forma como são descritos os gráficos em duas ou mais ferramentas de visualização podem ser diferentes.

A conversão dos eventos é uma forma de se realizar o mapeamento dos eventos necessário para a visualização. Essa conversão é feita normalmente baseada no tipo do evento de entrada. Essa conversão deve utilizar as informações registradas no evento para que possam ser gerados os respectivos eventos da ferramenta de visualização, de acordo com a visualização desejada. Em algumas situações, um determinado evento pode gerar múltiplos eventos que juntos conseguem obter a visualização desejada em uma ferramenta de visualização. Por exemplo, um evento de troca de estado de um processo pode ser mapeado para eventos de criação de barras horizontais que representam processos na visualização, além da troca de cores dessas barras e um evento pontual registrando o início desse novo estado. O processo de conversão dos eventos de entrada deve levar em conta a unificação dos identificadores assim como a hierarquia unificada descritas nas seções anteriores.

Como a conversão dos dados de uma determinada ferramenta de coleta está ligada aos tipos dos eventos que essa ferramenta registra, convém separar a conversão dos eventos em componentes independentes. Dessa forma, cada componente mantém a conversão acerca da semântica dos eventos e permite que, caso se altere a forma de visualização de uma determinada ferramenta, seja alterado somente o componente de conversão correspondente. Entretanto, a divisão em componentes deve utilizar os identificadores unificados e a forma de visualização integrada dos dados. A Figura 3.6 ilustra a divisão do processo de padronização em dois componentes de conversão, um para os dados de cada ferramenta de coleta utilizadas na primeira etapa.

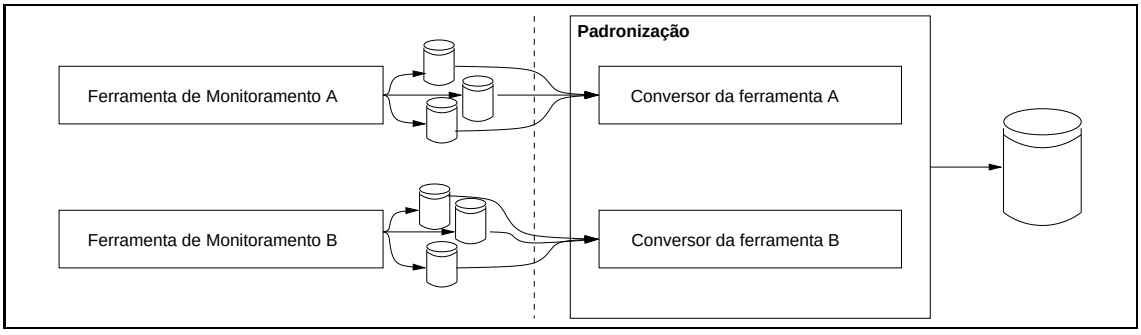

Figura 3.6: Figura mostrado o processo de padronização dos dados utilizando um conversor para cada ferramenta de monitoramento ou biblioteca de rastreamento

3.4 Conclusão do capítulo

Neste capítulo foi apresentado o modelo de integração de informações coletadas por diferentes ferramentas de monitoramento. Foi visto que o modelo é dividido basicamente em duas etapas: a da coleta de dados, que é feita pelo período que se deseja monitorar as informações; e da integração dos dados, onde as informações são tratadas de forma que possam ser visualizadas conjuntamente.

No próximo capítulo, de implementação, será descrito o protótipo de integração de dados construído neste trabalho. Essa descrição inicia através da apresentação das ferramentas de coleta utilizadas e termina com a descrição das decisões de implementação tomadas para que a integração de acordo com o modelo deste capítulo fosse feita.

4 IMPLEMENTAÇÃO DO MODELO

No capítulo anterior foi apresentado o modelo do sistema proposto para resolver os problemas mostrados no capítulo 2. Esse modelo, baseado na depuração *offline*, divide o processo de visualização e análise de informações em duas etapas: coleta e integração das informações.

Neste capítulo será descrita a implementação do modelo. A figura 4.1 mostra as fontes de dados utilizadas no trabalho: rastros de aplicações desenvolvidas com DECK e MPI, informações do Ganglia e do Performance Co-Pilot e dados de rastreamento de processos do Sistema Operacional. O resultado da coleta de dados por essas fontes de informação é passado ao protótipo de integração de dados, como pode ser visto também nessa figura. As fontes de dados serão descritas na seção 4.1 enquanto que a construção do protótipo será descrita na seção 4.2.

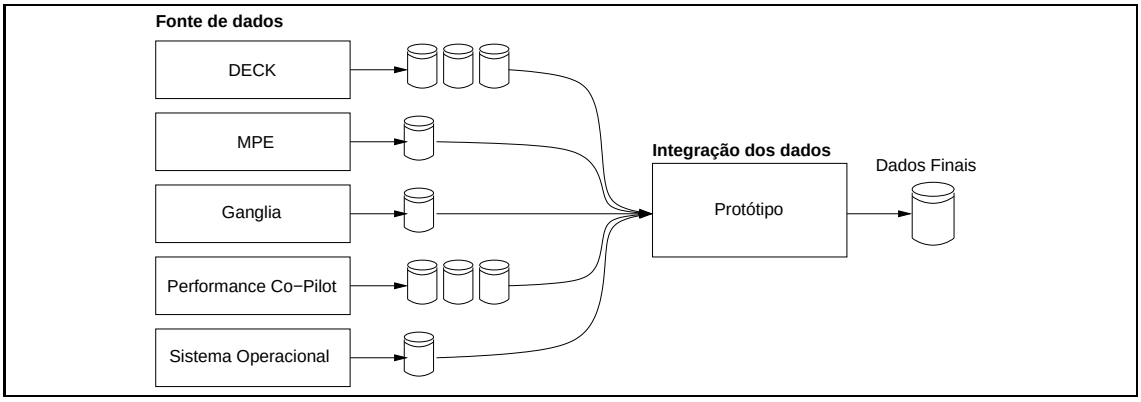

Figura 4.1: Fontes de dados utilizadas e o protótipo construído para a implementação do modelo de integração de dados

4.1 Coletores

Nesse trabalho foram utilizadas cinco sistemas de coleta, entre os quais três já existentes: Ganglia, Performance Co-Pilot e MPE. Os dois primeiros são utilizados por administradores de *cluster* como ferramentas de monitoramento. O terceiro, MPE, é uma biblioteca de rastreamento para a depuração de programas MPI. Os outros dois sistemas, construídos neste trabalho, são o rastreamento de aplicações DECK e rastreamento de processos no nível do sistema operacional. Com esse conjunto de sistemas de coleta de dados pretende-se englobar tanto a área de depuração de aplicações paralelas quanto a

área de administração de *cluster*. Nesta seção serão descritos cada um desses sistemas em mais detalhes.

4.1.1 DECK

O DECK (BARRETO; RIVIERE; NAVAUX, 1998) é um ambiente de desenvolvimento de aplicações paralelas. Esse ambiente é composto por um sistema de execução, uma biblioteca de comunicação e funções para gerenciamento de fluxos de execução. Nesta seção serão descritas as informações que se deseja obter de uma aplicação DECK quando executada, como foi feito o registro dessas informações e, no fim, a hierarquia de entidades e os identificadores utilizados para cada uma das entidades. No apêndice A.1 pode-se encontrar mais detalhes sobre o DECK.

Uma forma de se realizar a depuração de aplicações DECK é através do registro de quando determinadas funções foram executadas e por quanto tempo. Isso é o que se deseja obter de informação no rastreamento de aplicações desenvolvidas com essa biblioteca.

Para realizar o rastreamento das funções DECK, utilizou-se a técnica de instrumentação. Essa instrumentação consiste em encontrar pontos de rastreamento no código das funções e colocar ali um conjunto de chamadas que seja capaz de registrar informações no momento que esse ponto de rastreamento é executado.

Tabela 4.1: Funções da biblioteca DECK instrumentadas e seus pontos de instrumentação. Os pontos de instrumentação escritos com IN são normalmente no início das funções, enquanto que os escritos com OUT são no fim. Os pontos de instrumentação únicos para uma determinada função são normalmente no início da função, com exceção do ponto de instrumentação DECK_MBOX_POST.

Função DECK	Pontos de Instrumentação
`deck_thread_create`	DECK_THREAD_CREATE
`deck_thread_join`	DECK_THREAD_JOIN_IN, DECK_THREAD_JOIN_OUT
`deck_thread_yield`	DECK_THREAD_YIELD_IN, DECK_THREAD_YIELD_OUT
`deck_thread_sleep`	DECK_THREAD_SLEEP_IN, DECK_THREAD_SLEEP_OUT
`deck_thread_usleep`	DECK_THREAD_USLEEP_IN, DECK_THREAD_USLEEP_OUT
`deck_thread_barrier`	DECK_THREAD_BARRIER_IN, DECK_THREAD_BARRIER_OUT
`deck_mbox_create`	DECK_MBOX_CREATE
`deck_mbox_clone`	DECK_MBOX_CLONE_IN, DECK_MBOX_CLONE_OUT
`deck_mbox_destroy`	DECK_MBOX_DESTROY
`deck_mbox_post`	DECK_MBOX_POST
`deck_mbox_retrv`	DECK_MBOX_RETRV_IN, DECK_MBOX_RETRV_OUT
`deck_group_create`	DECK_GROUP_CREATE_IN, DECK_GROUP_CREATE_OUT
`deck_group_destroy`	DECK_GROUP_DESTROY
`deck_collective_msgbcast`	DECK_GROUP_MSGBCAST_IN, DECK_GROUP_MSGBCAST_OUT
`deck_collective_bcast`	DECK_GROUP_BCAST_IN, DECK_GROUP_BCAST_OUT
`deck_collective_barrier`	DECK_GROUP_BARRIER_IN, DECK_GROUP_BARRIER_OUT
`deck_collective_scatter`	DECK_GROUP_SCATTER_IN, DECK_GROUP_SCATTER_OUT
`deck_collective_gather`	DECK_GROUP_GATHER_IN, DECK_GROUP_GATHER_OUT
`deck_collective_reduce`	DECK_GROUP_REDUCE_IN, DECK_GROUP_REDUCE_OUT

Essa técnica foi utilizada para se instrumentar as funções da biblioteca listadas na Tabela 4.1. Essas funções são responsáveis pela comunicação através de caixas de mensagens, pelo gerenciamento de fluxos de execução e pela comunicação coletiva (CASSALI et al., 2000). Em cada um dos pontos de instrumentação, é gerado um evento que contém um determinado tipo e informações relacionados a esse tipo. A tabela mostra ainda a especificação dos pontos de instrumentação detectados. Por exemplo, a função `deck_thread_barrier` tem como objetivo sincronizar todos os fluxos de execução de um processo. Para saber por quanto tempo os fluxos ficaram bloqueados nessa rotina, registra-se no início da função um evento de início de barreira, com a data, identificações do fluxo de execução e do processo. O evento de tipo fim da barreira é registrado no fim da execução dessa função com as mesmas informações do evento de início da barreira. Com essas informações, é possível saber então por quanto tempo um determinado fluxo de execução ficou bloqueado na barreira. A implementação dos pontos de instrumentação das outras funções é semelhante a esse exemplo, com exceção das funções que realizam comunicação, onde informações adicionais, em pontos de instrumentação extras, tiveram de ser adicionados.

A implementação dos pontos de instrumentação detectados foi feita utilizando-se a biblioteca libRastro, descrita no apêndice A.4. A utilização dessa biblioteca de instrumentação tem uma baixa sobrecarga, já que utiliza *buffers* em memória e tem um formato de arquivo binário pequeno (SILVA; OLIVEIRA STEIN, 2002). Através dela é possível obter informações que podem ser utilizadas para a sincronização dos rastros gerados por diferentes processos em diferentes máquinas de um *cluster*. A Figura 4.2 ilustra o funcionamento do rastreamento em uma aplicação desenvolvida com o DECK instrumentado. Cada fluxo de execução gera um arquivo com os seus rastros. O resultado do rastreamento, ilustrado na figura pelos rastros de execução, pode ser lido após a execução da aplicação com funções da libRastro.

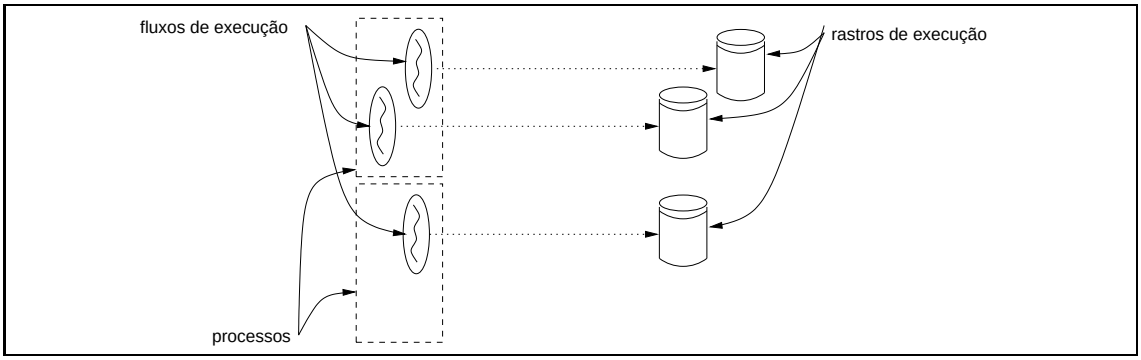

Figura 4.2: Funcionamento do rastreamento de uma aplicação utilizando a biblioteca DECK instrumentada com a libRastro

Mesmo com a baixa sobrecarga proporcionada pela utilização da libRastro, a instrumentação feita na biblioteca procurou ser pouco intrusiva em termos de desempenho durante a execução e em termos de facilidade de compilação e isolamento dos pontos de instrumentação. Isso significa que se pode desabilitar a instrumentação antes da compilação da biblioteca DECK, caso se queira uma biblioteca não instrumentada. Caso se opte por compilar a biblioteca com a instrumentação, o programador tem a possibilidade de habilitar e desabilitar o rastreamento em seu programa através da utilização de um conjunto específico de funções para esse fim. Através dessas funções é possível também

rastrear somente um conjunto específicos das funções instrumentadas. Isso permite ao programador uma maior flexibilidade na utilização do rastreamento e diminui a intrusão da instrumentação. Alguns resultados preliminares da instrumentação e visualização de dados da biblioteca DECK utilizando o rastreamento da libRastro foram publicados em SCHNORR; STEIN; NAVAUX (2004).

Hierarquia e Identificadores

A Figura 4.3 ilustra a hierarquia de entidades obtida com o rastreamento da biblioteca DECK através da técnica de instrumentação. Nessa hierarquia, a aplicação DECK pode ser executada em várias máquinas, cada uma podendo ter vários fluxos de execução. Cada fluxo de execução pode estar em um dos estados representados na figura. Existe um estado para cada função DECK, além daquele que indica a não execução de funções da biblioteca.

Os identificadores utilizados no rastreamento das funções DECK são os números de processos para os fluxos de execução, os nomes das máquinas para os nós que são utilizados para executar esses fluxos de execução e um número definindo cada tipo de estado que um fluxo de execução pode ter. Essas informações sobre a hierarquia de entidades e os identificadores utilizados serão necessários posteriormente na integração dos rastros de execução de aplicações DECK com dados registrados por outras ferramentas.

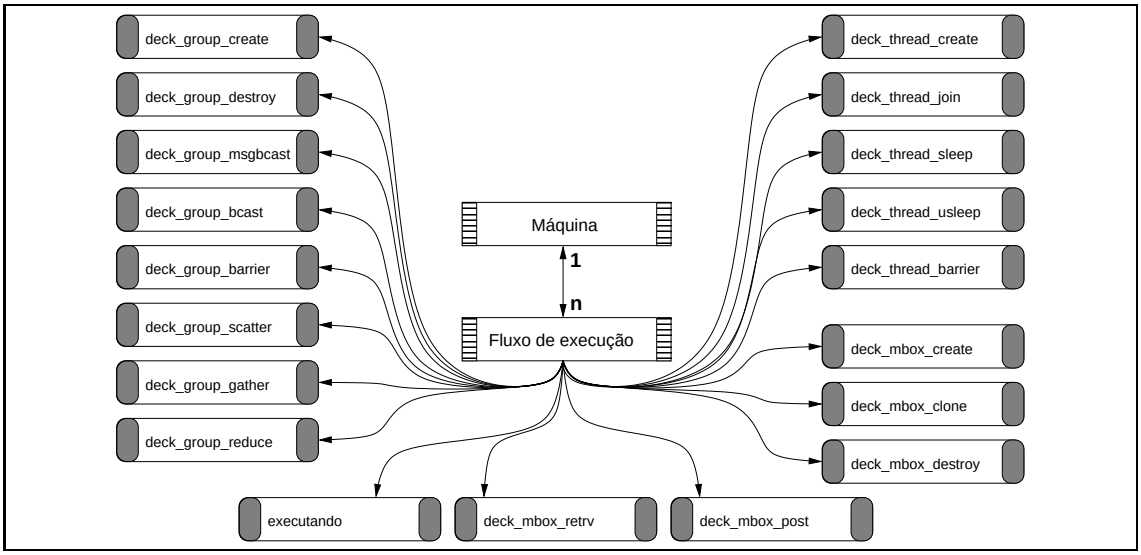

Figura 4.3: Hierarquia de entidades do sistemas de rastreamento construído através da técnica de instrumentação da biblioteca DECK

4.1.2 MPI

MPI (*Message Passing Interface*) (WALKER; DONGARRA, 1996) é um padrão para a implementação de bibliotecas para passagem de mensagem. As diversas implementações existentes e o padrão bem definido tornam a sua utilização bem difundida no desenvolvimento de aplicações paralelas. O rastreamento de aplicações MPI pode ser feito através da biblioteca MPE (*Multiprocessing Parallel Environment*), que já é distribuída com a implementação MPICH (GROPP et al., 1996). Nesta seção será descrito o que essa biblioteca de rastreamento oferece em termos de registro de comportamento e como ela realiza a captura dos eventos. No final da seção é descrita a hierarquia de entidades

das informações geradas pelo rastreador MPI e os identificadores utilizados para essas entidades.

Aplicações paralelas se caracterizam por ter períodos de comunicação e períodos de processamento. A biblioteca de rastreamento MPE registra quando um processo MPI está bloqueado realizando comunicação. Como a biblioteca não identifica explicitamente quando o código do programador é executado, toma-se por base que essas regiões são executadas quando não se está executando nenhuma rotina de comunicação. Essa biblioteca identifica também por quanto tempo as comunicações duraram e com quem foi feita a comunicação. A Figura 4.4 mostra uma possível visualização do comportamento de uma aplicação paralela baseado nas informações que são registradas pela biblioteca MPE. Nessa figura, são mostrados os diferentes estados que um processo MPI pode ter: regiões "A" para execução do código do programador; região "B" para rotinas bloqueantes de recebimento e a região "C" para rotinas bloqueantes de envio. A figura mostra além disso uma comunicação entre os processos 1 e 0.

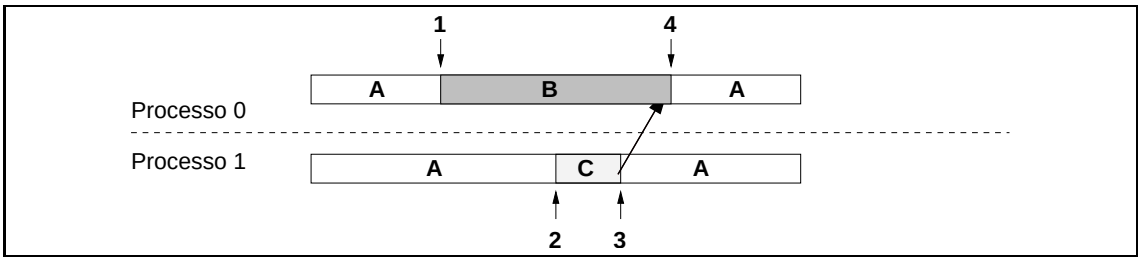

Figura 4.4: Diferentes estados de processos MPI e a comunicação entre dois desses processos

A Figura 4.5 mostra a forma de funcionamento da biblioteca MPE integrada a uma aplicação MPI. Nessa figura, a função `recebe_dados` é construída pelo programador, enquanto que a `MPI_Recv` é uma função da biblioteca MPI. A biblioteca MPE intercepta a chamada a função `MPI_Recv`, registrando o evento de troca de estado para bloqueado e aguardando dados. Esse evento contém o número do processo MPI e a data quando aconteceu. Após o registro desse evento, a interceptação termina e a função `MPI_Recv` é chamada normalmente. O retorno da função `MPI_Recv` é também interceptado, registrando neste momento outro evento indicando fim do estado bloqueado e a recepção de uma mensagem. Após isso, o código do programador volta a ser executado. Esse exemplo ilustra o comportamento da biblioteca MPE integrada com o processo MPI de número 0 da Figura 4.4, sendo que a visualização gráfica entre os momentos 1 e 4 representa a execução da função `MPI_Recv`.

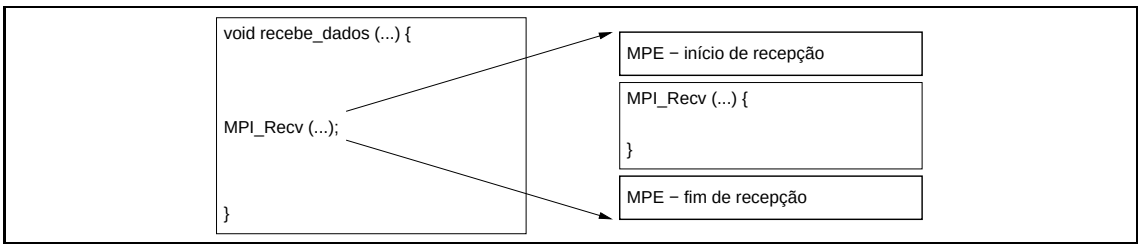

Figura 4.5: Funcionamento da biblioteca MPE integrada com uma aplicação MPI com uma função recebe_dados que executa MPI_Recv

A biblioteca MPE, além de capturar chamadas as rotinas de comunicação MPI, permite ao próprio programador instrumentar sua aplicação. As informações que são registradas pela instrumentação do programador são unidas com as informações registradas pela MPE durante a execução da aplicação. Outra característica da biblioteca MPE é a sua utilização ser independente do código do desenvolvedor MPI. Dessa forma, o programador não precisa alterar a sua aplicação para utilizar o rastreamento que essa biblioteca oferece. No momento da ligação da aplicação MPI com essa biblioteca, as chamadas as funções de comunicação são substituídas pelas funções da MPE correspondentes. Essas funções correspondentes implementam o rastreamento e depois chamam a função principal que realiza a comunicação em si.

O resultado do rastreamento de uma aplicação MPI com a biblioteca MPE é um arquivo com eventos ordenados temporalmente. Cada evento desse arquivo tem um tipo e outras informações relacionadas a esse tipo. Dependendo do tipo do evento, ele pode descrever diferentes trocas de estado da aplicação MPI. A Tabela 4.2 mostra alguns eventos de um arquivo de rastreamento. Cada evento foi gerado em determinado processo, indicado pelo campo `pid`, e tem um tipo, indicado pela combinação dos campos `type` e `id`. O tempo é registrado nos campos `timestamp`. O campo `data` é utilizado naqueles eventos que indicam uma comunicação entre processos. O valor deste campo indica com qual processo houve a comunicação.

Tabela 4.2: Composição de eventos e o respectivo significado de registros encontrados em um arquivo gerado pela biblioteca MPE de rastreamento.

Composição do evento					
`timestamp`	`type`	`id`	`pid`	`data`	Significado do evento
0.004514	loc		0		criação de um processo MPI
0.004812	loc		1		criação de um processo MPI
0.276475	raw	161	0	1	envio de dados
0.276777	raw	-102	1	0	recepção de dados

O rastro gerado pela biblioteca MPE em conjunto com a biblioteca MPI tem entre suas características duas especialmente importantes: a forma como é registrada a data dos eventos, sendo uma data relativa ao início da execução, e a identificação dos processos MPI, feita por números seqüenciais distribuídos de acordo com o arquivo de configuração de nós do ambiente de execução MPI. Essas duas características geram dificuldades para realizar a integração dos eventos com outros oriundos de outras fontes. Para facilitar a integração desses eventos foi feito um código que é adicionado ao código do programador automaticamente em tempo de compilação. Esse código utiliza definições do pré-processador da linguagem C para alterar a chamada `MPI_Init`. O código adicionado pelo pré-processador registra a data local de início da execução da aplicação e registra, para cada processo MPI, qual o número do processo utilizado pelo sistema operacional. Com isto, o programador não precisar alterar o código de sua aplicação e é possível realizar posteriormente a integração desses eventos MPI com eventos gerados por outras ferramentas de rastreamento.

A biblioteca MPE de rastreamento tem seu código aberto mas somente é distribuída com uma das várias implementações do padrão MPI. No entanto, algumas adaptações podem ser feitas para que ela funcione com outras implementações MPI. Essas adaptações, em sua maioria, consistem em alterar a forma de compilação da aplicação MPI desenvol-

vida.

Hierarquia e Identificadores

A Figura 4.6 mostra a a hierarquia de entidades de uma aplicação MPI. Um programa MPI pode conter vários processos. Cada um desses pode estar em três estados diferentes: enviando, recebendo e executando.

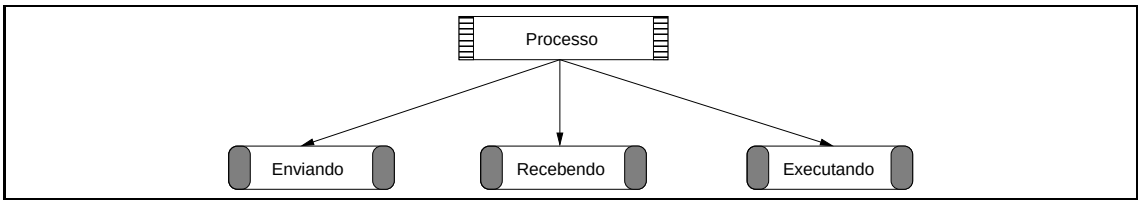

Figura 4.6: Hierarquia de entidades da uma aplicação MPI: cada aplicação MPI é composta por um conjunto de processos MPI sendo que cada processo pode estar no estado enviando, recebendo e executando.

Os identificadores da entidade "processo" são os números seqüenciais distribuídos de acordo com o arquivo que diz em quais máquinas a aplicação MPI deve ser executada. No caso dos estados, é utilizado um número constante para cada estado possível de um processo MPI. As informações hierárquicas e de identificadores são necessárias para a etapa de integração das informações de rastreamento do MPI com dados oriundos de outras ferramentas de monitoramento.

4.1.3 Ganglia

Ganglia (MASSIE; CHUN; CULLER, 2004) é uma ferramenta de monitoramento de *clusters*. Ela é capaz de coletar informações de cada máquina de um *cluster*, como a quantidade de tempo de processamento utilizada em um determinado momento, espaço livre e utilizado em disco, memória que está sendo utilizada. Nesta seção serão descritas as informações que se deseja capturar do Ganglia e a forma como foi feita essa captura, primeiramente mostrando o funcionamento normal da ferramenta e depois explicando a construção de uma aplicação auxiliar de coleta. No final, é apresentada a hierarquia de entidades geradas pelo rastreamento dos dados coletados pelo Ganglia e os respectivos identificadores. Mais detalhes acerca dessa ferramenta podem ser encontrados no apêndice A.2.

As informações que são obtidas pelo Ganglia, como as citadas anteriormente, podem ajudar um desenvolvedor de aplicação paralela a entender o comportamento de seu programa paralelo (KARAVANIC et al., 1997). Além disso, essas informações podem permitir ao administrador do *cluster* ter um maior controle no que está disponível aos usuários do *cluster* e em que nível está essa disponibilidade (SACERDOTI; MASSIE; CULLER, 2003). São essas informações que se deseja obter da ferramenta de forma que sejam integradas posteriormente na segunda fase do processo de integração.

A Figura 4.7 mostra o funcionamento da ferramenta de monitoramento Ganglia. O sistema contém um agente monitor, chamado `gmond`, que realiza a coleta de informações em cada nó do *cluster*. Esse agente é responsável por coletar as informações locais (flechas A, figura 4.7) e periodicamente consultar os outros agentes através de um canal multicast (flechas B, figura 4.7). Essa consulta tem por objetivo fazer com que cada monitor mantenha, em um determinado momento, dados de todos os nós do *cluster* (flechas

C, figura 4.7). As informações dos agentes são mantidas em memória. Isso permite um maior desempenho e uma menor intrusão quando esses monitores são utilizados. Além da conexão multicast existente entre os agentes, cada monitor mantém um canal de comunicação através do qual uma aplicação externa pode obter as informações mantidas pelo agente.

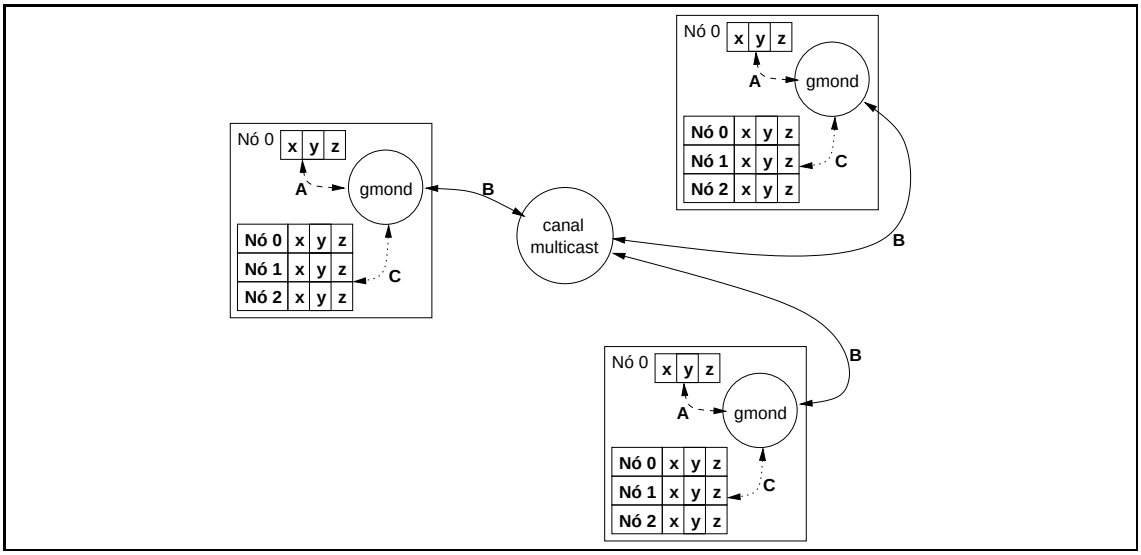

Figura 4.7: Estrutura de coleta de informações do Ganglia

As informações que são coletadas pelo Ganglia são trocadas entre os agentes `gmond`‘s através da rede de interconexão e mantidas em memória. Essas informações são registradas em arquivo por outro tipo de agente, chamado `gmetad`. As informações coletadas pelo `gmetad` são registradas em bases de dados circulares. Essa forma de registro não permite a utilização do monitoramento por períodos longos mantendo uma freqüência significativa de coletas. Além disso, coletar dados dessas bases de dados circulares é um processo difícil de ser controlado pois o monitoramento através do `gmetad` implica que a coleta deve estar sempre sendo feita. Foi construído neste trabalho uma aplicação auxiliar para resolver esses problemas e ainda permitir o registro dos dados coletados pelos agentes `gmond`. Essa aplicação pode ser controlada pelo usuário e permite ainda a filtragem de quais informações devem ser registradas.

Essa aplicação auxiliar de registro dos dados obtém as informações de um determinado agente `gmond` através da conexão na porta de comunicação desse agente. A aplicação é configurável e tem a possibilidade de filtrar quais dados se deseja rastrear, além de quais máquinas se quer monitorar. A Figura 4.8 ilustra o funcionamento da aplicação construída quando integrada ao sistema de coleta do Ganglia. Em um momento inicial, no passo 1, a aplicação recebe um arquivo de configuração dizendo de quais máquinas ele deve registrar dados e quais as informações que deverão ser monitoradas. No passo 2, a aplicação rastreadora se conecta com algum dos monitores locais do *cluster* e, no passo 3, obtém os dados. A aplicação rastreadora, nesse momento, realiza a filtragem das informações de acordo com os arquivos de configuração e então registra os dados relevantes em arquivo, no passo 4. Os passos 2, 3 e 4 acontecem periodicamente.

O resultado do processo de rastreamento dos dados coletados pelo Ganglia é um arquivo com os dados coletados. A Tabela 4.3 mostra alguns dados de um desses arquivos. Cada linha representa o rastreamento de uma determinada informação em determinado

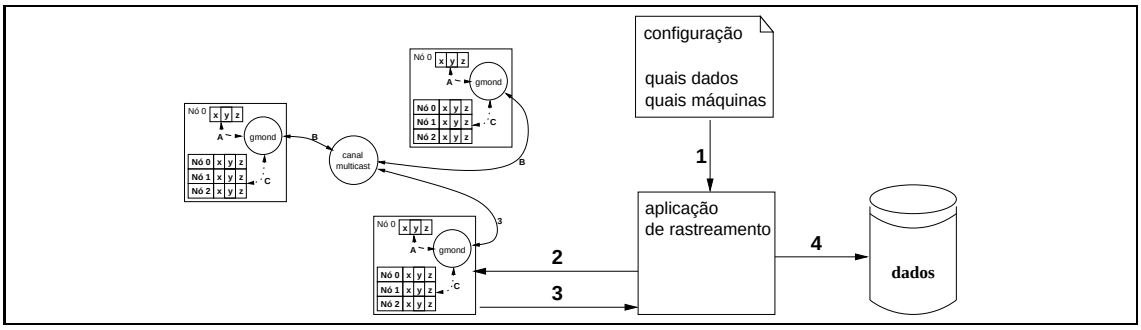

Figura 4.8: Funcionamento do sistema de rastreamento de informações coletadas pelo Ganglia

instante de tempo. As duas primeiras colunas mostram a data e o tipo da informação, respectivamente. A terceira coluna é a qual máquina se refere aquela informação e a última coluna é o valor daquele tipo de informação.

Tabela 4.3: Composição de registros de um arquivo gerado pelo rastreamento dos dados coletados pelo Ganglia

`data`	`tipo`	`máquina`	`valor` (Kbytes)
1096944667	bytes_out	athlon01.lsc.ufsm.br	0.59
1096944667	bytes_out	paple01.lsc.ufsm.br	7868250.00
1096944667	bytes_out	paple02.lsc.ufsm.br	7769665.50
1096944668	bytes_out	athlon02.lsc.ufsm.br	0.03

Hierarquia e Identificadores

A Figura 4.9 mostra a hierarquia de entidades simplificada das informações coletadas pelo sistema descrito nesta seção. Em linhas gerais, para cada máquina do *cluster*, tem-se um conjunto de informações que podem ser rastreadas pelo sistema construído. Na figura, os tipos de informação podem ser dados de utilização do processador, memória e placa de rede. Dependendo de como é feita a configuração do rastreador e as informações coletadas pelo Ganglia, outras entidades podem ser adicionadas nesse nível.

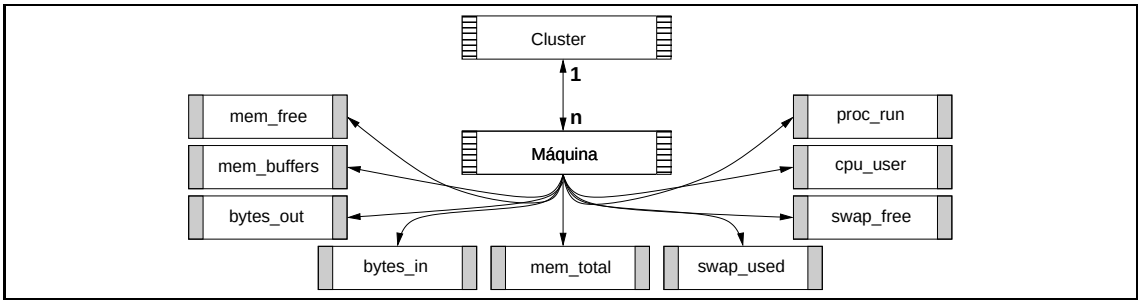

Figura 4.9: Hierarquia de entidades gerados pelo rastreamento de informações coletadas pelo Ganglia

Para identificar as máquinas que fazem parte da hierarquia de entidades do Ganglia

são utilizados os nomes das máquinas. As informações são identificadas através de *strings* correspondentes à determinada informação. Por exemplo, `cpu_user` é o identificador do valor da informação que se refere a utilização da CPU pelo usuário.

4.1.4 Performance Co-Pilot

Performance Co-Pilot (PCP) (SGI, 1999) é uma ferramenta de monitoramento para *cluster*. Ela é composta por um conjunto de programas que realizam a coleta e o registro dos dados coletados em arquivo. Além disso, possui uma biblioteca que pode ser utilizada por novos programas para adicionar novas métricas a serem monitoradas e para acessar os dados monitorados. Nesta seção serão descritos os dados disponibilizados por essa ferramenta, como foi feita a coleta dos dados, a hierarquia de suas entidades e os identificadores dessas entidades. Mais detalhes sobre essa ferramenta estão no apêndice A.3.

As informações que o Performance Co-Pilot disponibiliza são semelhantes àquelas coletadas pelo Ganglia. Essas informações são, por exemplo, a carga dos processadores, dados do sistema de arquivos da rede, utilização da placa de rede, memória e disco, quantidade de interrupções do processador, entre outras. No entanto, o Performance Co-Pilot tem um conjunto maior de dados que podem ser coletados. As informações que serão disponibilizadas são aquelas que vem em uma instalação padrão do PCP.

A ferramenta PCP pode ser organizada de diferentes formas através da combinação de seus componentes. A Figura 4.10 ilustra a organização do sistema do Performance Co-Pilot utilizada neste trabalho. Cada nó do *cluster* é monitorado por um agente, chamado `pmcd`, que coleta dados locais e mantém as informações hierarquicamente organizadas em memória. Esse agente é dividido em módulos chamados `pmda`, sendo que cada um desses módulos fica responsável por coletar um determinado conjunto de métricas. O `pmlogger` é um programa responsável pelo registro das informações em arquivo. Esse programa se conecta ao agente `pmcd`, obtém os dados e os registra em arquivo. A freqüência dessa conexão depende de como é configurado o `pmlogger` e quão precisa será a métrica arquivada. A configurabilidade da ferramenta permite, por exemplo, coletar um conjunto de métricas de 1 em 1 segundo e outro conjunto, de 10 em 10 minutos.

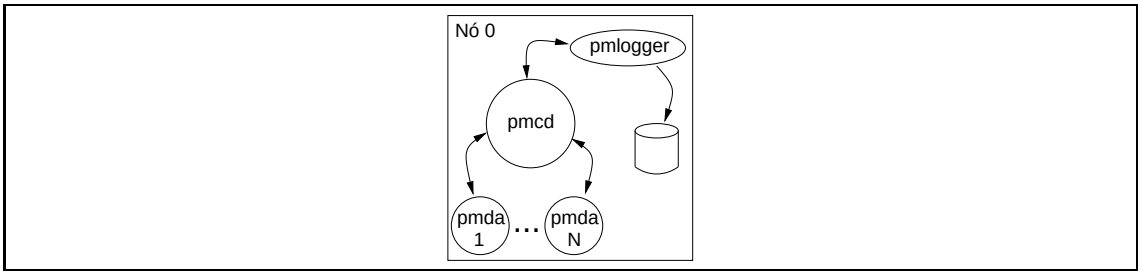

Figura 4.10: Organização por máquina do sistema PCP utilizado nesse trabalho

O resultado da coleta de dados pelo Performance Co-Pilot é um arquivo para cada máquina monitorada. Esse arquivo é dividido em registros ordenados temporalmente, sendo que cada registro contém o dado que foi coletado, assim como o seu identificador. A freqüência do registro de determinado tipo varia de acordo com a freqüência com que esse tipo de informação foi coletada. A Tabela 4.4 mostra alguns registros gerados pelo `pmlogger`. Esses registros tem informação de monitoramento da utilização da CPU (kernel.all.load) coletados pelos módulo `pmda` kernel do `pmcd` de 1 em 1 segundo e dados de

memória livre (mem.util.free) coletados pelo pmda “mem” do pmcd. Os identificadores dessas informações, ou seja, kernel.all.load e mem.util.free representam a organização hierárquica das informações.

Tabela 4.4: Composição simplificada de registros de um arquivo gerado pelo pmlogger da ferramenta de monitoramento para *cluster* Performance Co-Pilot

data	identificação	valor
63	kernel.all.load	0.5
63	mem.util.free	345928
64	kernel.all.load	0.4
65	kernel.all.load	0.2
66	kernel.all.load	0.3
67	kernel.all.load	0.2
68	kernel.all.load	0.5
68	mem.util.free	340587

Hierarquia e Identificadores

A Figura 4.11 mostra a hierarquia de entidades simplificada do sistema de rastreamento do Performance Co-Pilot. Cada *cluster* é composto por máquinas e cada máquina contém uma hierarquia de entidades onde as sub-árvores são definidas por quais módulos do pmcd estão sendo utilizados. Na figura, os módulos pmda utilizados são o kernel, mem, swap e network. Cada módulo pmda contém uma sub-hierarquia de entidades, como pode ser visto na figura.

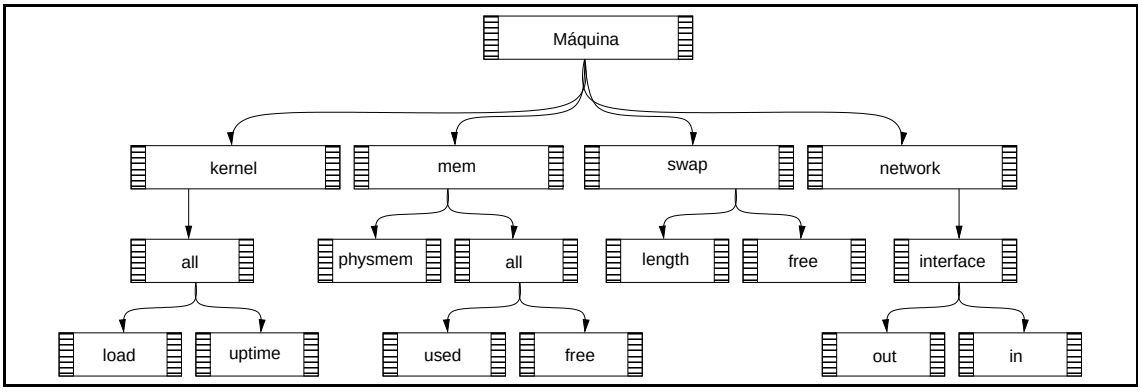

Figura 4.11: Hierarquia de entidades do sistema de coleta de dados do Performance Co-Pilot

O Performance Co-Pilot utiliza os nomes dos nós como identificador da entidade “máquina” e, para cada entidade filha dessa entidade máquina, o próprio nome como identificador. Por exemplo, para a entidade “kernel”, o identificador é o próprio nome da entidade, ou seja, kernel.

4.1.5 Sistema Operacional

O sistema operacional é quem gerencia os recursos das máquinas locais em um *cluster*. Esse gerenciamento se dá através da políticas de alocação desses recursos e controle

dos usuários. O sistema operacional utilizado neste trabalho é o Linux. Nesta seção serão vistas as informações que se deseja obter do núcleo do Linux em relação ao escalonamento e a forma como foi feita a obtenção dessas informações. No final, é apresentada a hierarquia de entidades para o sistema de rastreamento construído e os identificadores das entidades utilizados.

A política de escalonamento do sistema operacional influencia diretamente o desempenho de um determinado processo em uma máquina. Realizar o rastreamento da política de escalonamento, registrando quando um determinado processo vai ser executado e quando é bloqueado pelo sistema operacional é uma forma de fazer o monitoramento dessa política. O que se quer de informação do sistema operacional são as trocas de contexto que sofre um processo e seus fluxos de execução.

Existem soluções que permitem o rastreamento de processos no nível do sistema operacional como *FKT - Fast Kernel Tracing* (RUSSELL; CHAVAN, 2002), VPPB (BROBERG; LUNDBERG; GRAHN, 1999) e *LTT - Linux Trace Toolkit* (YAGHMOUR; DAGENAIS, 2000). Estas soluções permitem o rastreamento da troca de contexto de processos e um controle na utilização e alocação de tamanho de buffers de registro. No entanto, a implementação da primeira delas, FKT, está somente disponível para uma versão antiga do núcleo do Linux. No caso da segunda, VPPB, a implementação somente existe para o Sistema operacional Solaris. A terceira, LTT, está disponível para versões atuais do núcleo do sistema operacional Linux mas não tem uma documentação acerca do formato do arquivo gerado, dificultando a integração dos seus dados com informações de outras ferramentas de coleta.

Foi implementado então uma alteração no núcleo do sistema operacional Linux para registrar o comportamento de processos. Essa implementação consistiu em encontrar pontos de instrumentação no código do escalonador do núcleo do sistema. Dois pontos de instrumentação foram detectados: a última rotina do núcleo que é executada antes de um processo ser colocado em execução e o ponto onde a execução recomeça dentro do sistema operacional após a fatia de tempo de execução de um processo. Em cada um desses pontos foi feito o registro de qual processo está mudando de estado e qual a data daquele momento.

O funcionamento do sistema de rastreamento de processos está ilustrado na Figura 4.12. Nessa figura, os pontos de instrumentação são mostrados no lado esquerdo e o sistema rastreador de processos está ilustrado do lado direito. Na inicialização do sistema de rastreamento, o usuário informa ao controlador do *buffer*, através da interface /proc do núcleo, quais os processos que devem ser monitorados. O controlador reinicia então o *buffer* onde ficam registradas as trocas de estado e habilita o rastreamento. A partir de então, cada vez que um dos pontos de instrumentação é executado, verifica-se qual é o processo em questão e caso for um que está sendo monitorado, um registro é feito no *buffer* circular na área direita da figura. Além disso, é verificado se o processo em questão é um fluxo de execução do número de processo monitorado. Caso for, também é registrado uma troca de estado no *buffer*.

O processo de rastreamento do kernel exige um *buffer* para que o registro das trocas de contexto sejam guardadas até que o usuário que está realizando o monitoramento obtenha as informações. Atualmente, tem-se um *buffer* circular de 400 mil registros cada um com 8 *bytes* de tamanho. O tamanho limitado de cada estrutura faz com que sejam registrados dois tipos de eventos: de tempo e de troca de estado. Os eventos de tempo são registrados a cada segundo e tem como informação qual o segundo atual no momento do monitoramento. O outro tipo de evento, de troca de estado, contém os micro-segundos

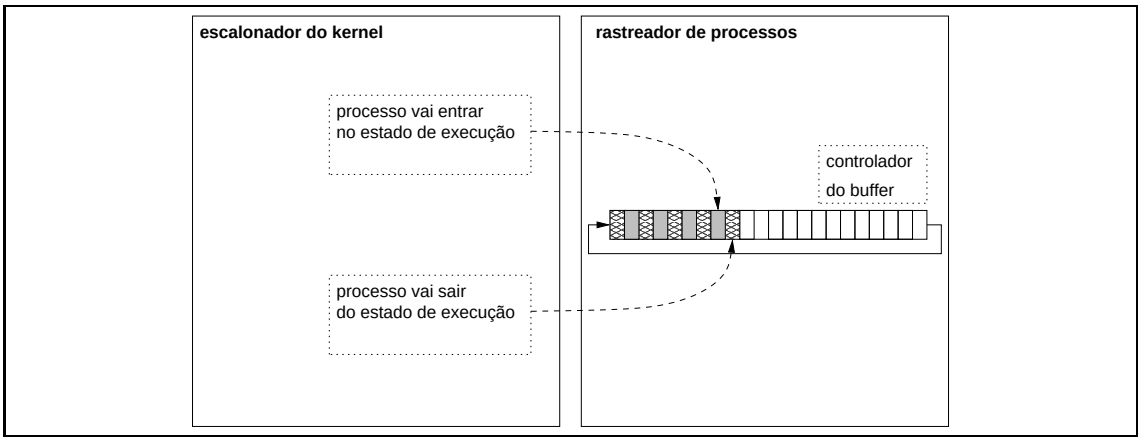

Figura 4.12: Funcionamento do rastreador de processos que funciona dentro do sistema operacional Linux

dentro do último segundo registrado, além do número de processo que trocou de estado, e qual o estado atual dele no momento do registro. Essa separação em dois tipos de eventos foi feita porque o tamanho necessário para guardar a data em segundos é relativamente grande se comparada aos dados úteis de cada evento.

Além dos controles usuais de configuração e inicialização através da interface /proc, o sistema de rastreamento permite que o registro das trocas de estado sejam suspensos temporariamente pelo usuário. Isso permite que o usuário possa obter dados de monitoramento somente durante a execução de uma determinada parte de sua aplicação.

Ao fim do processo de rastreamento, as informações são obtidas também através da interface /proc e gravadas em arquivo. A Tabela 4.5 mostra alguns dados extraídos de um desses arquivos. A primeira coluna da tabela mostra a ordem dos eventos em arquivo, a segunda coluna o tipo do evento, que pode ser de tempo, bloqueio ou desbloqueio, a terceira o tempo desse evento, que pode ser o segundo no caso de eventos do tipo tempo ou micro-segundos no caso de eventos de tipo bloqueio ou desbloqueio. A quarta e última coluna indica qual o processo em questão no momento daquele registro de troca de estado.

Tabela 4.5: Composição de registros de um arquivo gerado pelo rastreamento de processo implementado no núcleo do sistema operacional Linux

	`tipo`	`tempo`	`processo`
1	tempo	1096072237	-
2	desbloqueio	703861	23208
3	bloqueio	703869	23208
4	desbloqueio	703870	23208
5	bloqueio	703870	23208
6	desbloqueio	703870	23208
7	bloqueio	703871	23208
8	desbloqueio	703871	23208

Hierarquia e Identificadores

Na Figura 4.13 é mostrado a hierarquia de entidades do sistema de rastreamento criado. A entidade máquina pode conter um processo ou mais, caso seja uma aplicação com mais de um fluxo de execução. Cada processo pode estar no estado executando ou bloqueado.

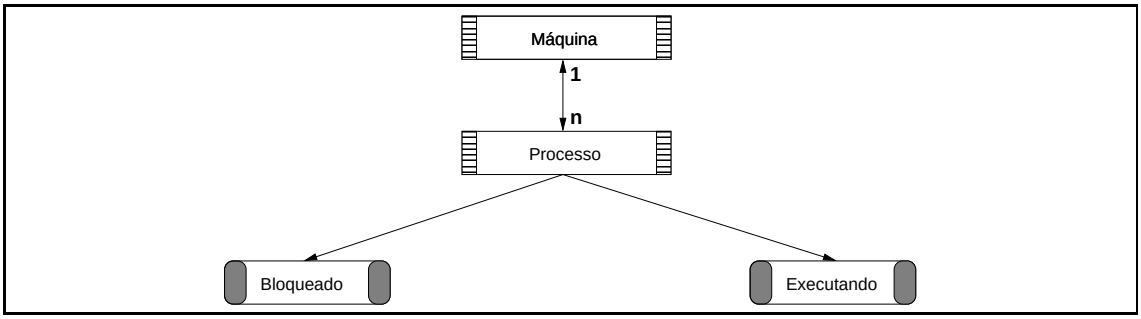

Figura 4.13: Hierarquia de entidades do sistema de rastreamento de processos

Em relação aos identificadores, as máquinas são identificadas através do seu nome enquanto que os processos são identificados através dos números de processo utilizados pelo sistema operacional. Os estados são identificadores através de dois valores constantes, um para cada estado.

4.2 Descrição do Protótipo

Na seção anterior foram descritos os coletores utilizados neste trabalho e que fazem parte da etapa de coleta de dados do modelo do capítulo 3. Esta seção mostra a implementação da segunda parte do modelo, através da descrição de uma ferramenta protótipo. Esta etapa consiste basicamente na integração das informações das diferentes ferramentas de coleta utilizadas anteriormente. O protótipo propõem-se a resolver as três questões principais do processo de integração de informação, que são: sincronização das informações, unificação das hierarquias e dos identificadores e padronização de formato. O produto final na utilização desta ferramenta protótipo é um arquivo capaz de ser visualizado na ferramenta Pajé (STEIN; KERGOMMEAUX; BERNARD, 2000).

O protótipo que será descrito nesta seção realiza a integração de dados oriundos dos rastreadores apresentados na seção anterior, que são: rastros de execução de aplicações DECK e MPI, dados coletados no sistema operacional Linux e informações registradas pelos monitoradores de *cluster* Ganglia e Performance Co-Pilot. Mesmo tendo utilizado apenas essas fontes de informação, a aplicação protótipo pode ser estendida de forma que outras fontes de dados possam ser utilizadas. O apêndice B.1 detalha o que deve ser feito para se integrar novos componentes ao protótipo.

A descrição da implementação do protótipo está organizada da seguinte forma. Inicialmente, a arquitetura e o funcionamento geral do protótipo serão mostrados de forma genérica, sem associar os seus componentes às fontes de informação utilizadas. Depois será descrito como foi feita a sincronização dos eventos registrados em arquivo, seguido pela descrição da unificação hierárquica das fontes de informação utilizadas. Após, serão mostrados a padronização de formato feita por componentes específicos do protótipo. No fim é apresentado o diagrama de classes do protótipo.

4.2.1 Arquitetura e Funcionamento geral do Protótipo

A arquitetura do protótipo está mostrada na Figura 4.14. Os principais componentes da ferramenta são as fontes de dados, o ordenador de dados, o controlador de integração e da aplicação. Os componentes fontes de dados são divididos em três sub-componentes: leitor das informações de arquivo (L), sincronizador (S) e conversor (C) dos eventos. Os arquivos que contêm a informação que foi registrada durante o período de monitoramento são lidos pelo leitor de dados dos componentes fonte de dados, sendo que cada uma dessas fontes de dados está relacionada a uma ferramenta de monitoramento ou biblioteca de rastreamento. O ordenador consulta as fontes de dados e obtém sempre o evento mais recente já padronizado, enviando este para o controlador de integração. O controlador de integração recebe as informações integradas do controlador de dados e as registra em arquivo. O controlador da aplicação inicializa o sistema através do controlador de integração e oferece ao usuário uma interface gráfica. A seguir é explicado mais detalhadamente cada um desses componentes. No final, a seqüência de funcionamento é apresentada.

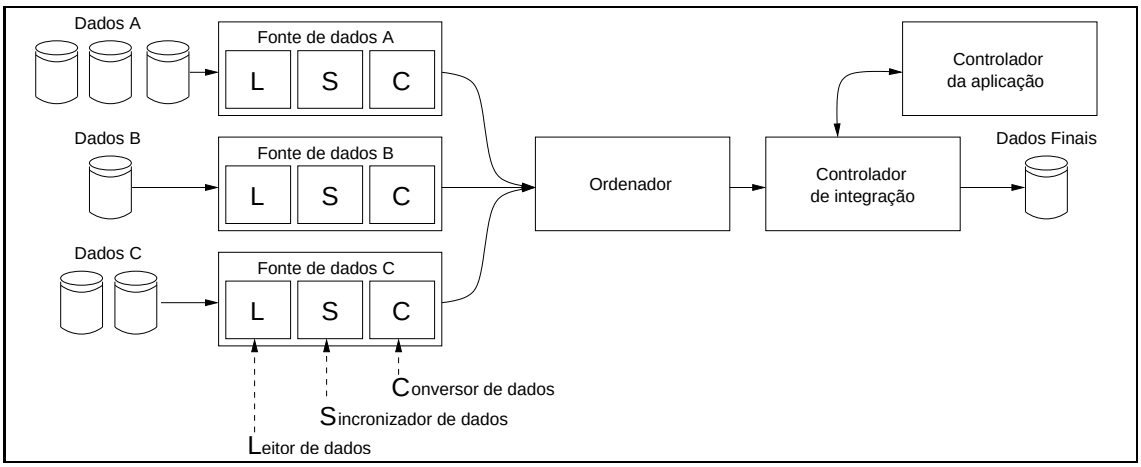

Figura 4.14: Arquitetura da ferramenta protótipo construída para implementar as tarefas da segunda parte do modelo de integração de dados

Fontes de Dados Os componentes fontes de dados são específicos a uma determinada ferramenta de monitoramento ou biblioteca de rastreamento. A Figura 4.15 mostra os sub-componentes de uma fonte de dados. As responsabilidades do componente fonte de informação são a sincronização das informações, utilização dos identificadores padronizados pelo controlador de integração e a padronização de formato dos eventos lidos. Os eventos lidos pelo leitor (L) tem seus tempos corrigidos no sincronizador (S) de acordo com um arquivo que contém informações de sincronização. A utilização dos identificadores e a padronização dos dados é feita pela sub-componente conversor (C) da fonte de dados. O resultados desse componente é um conjunto de eventos já convertidos para o formato de saída do protótipo.

Ordenador de dados O ordenador de dados gerencia os componentes fontes de dados. Seu objetivo é sempre ler o evento já convertido mais recente dentre os eventos disponíveis de todos os componentes fontes de dados que estiverem inicializados. O resultado dessa operação é passado ao controlador de integração.

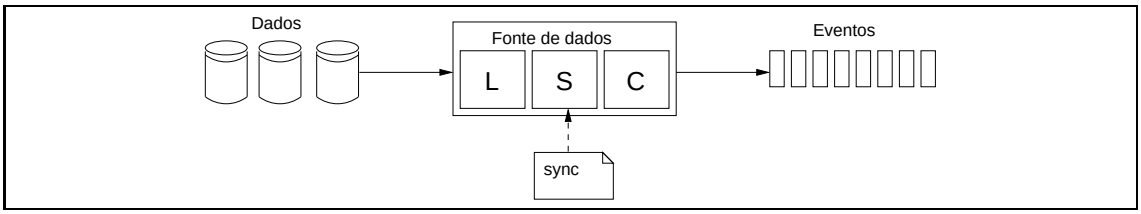

Figura 4.15: Arquitetura interna do componente fonte de dados com o leitor e o conversor de dados

Controlador de Integração O controlador de integração tem dois objetivos principais: o primeiro é fazer a unificação hierárquica dos dados e a segunda é fornecer aos conversores de dados das fontes de dados os identificadores das entidades por eles utilizadas. Além desses dois objetivos principais, o controlador de integração configura o sistema, baseado nos arquivos de configuração e de rastreamento passados pelo controlador de aplicação, e recebe do ordenador os eventos já convertidos, gravando-os em um arquivo único no formato Pajé.

Controlador de Aplicação O controlador de aplicação é responsável por oferecer ao usuário uma interface gráfica amigável para configuração do protótipo. Através dessa interface, o usuário diz quais são os arquivos que devem ser integrados e quais os arquivos de configuração devem ser passados aos componentes fonte de dados.

O funcionamento do protótipo é dividido em duas partes: a inicialização e depois a integração das informações. Na inicialização, o controlador de integração recebe do controlador da aplicação quais são os arquivos de dados que devem ser integrados e qual o formato desses arquivos. Com base nessas informações, o controlador de integração configura o ordenador de dados permitindo a inicialização de tantos componentes fontes de dados forem necessários para os arquivos que devem ser integrados. Os componentes fontes de dados recebem, através da inicialização do ordenador, o arquivo em que estão registradas as informações necessárias para a sincronização. Esse arquivo é utilizado pelo sincronizador (S) de cada fonte de dados. O controlador de integração recebe do controlador da aplicação o arquivo que descreve a unificação de dados que será realizada com as informações.

A Figura 4.16 mostra o funcionamento da integração de dados no protótipo, após a inicialização. Nessa figura, a flecha 1 representa a ordem do controlador de aplicação de iniciar a integração dos dados. A partir de então, na flecha 2, o controlador de integração requisita ao ordenador de dados o evento já convertido mais recente. O ordenador de dados verifica qual das fontes de dados tem o evento mais recente. Na flecha 3, o evento é requisitado e então o leitor daquele componente realiza a leitura do arquivo (L), a sincronização (S) do evento lido e a sua conversão (C). O evento lido, sincronizado e convertido é retornado ao ordenador de dados na flecha 4. Esse evento, recebido pelo ordenador, é repassado ao controlador de integração através da flecha 5. Esse controlador registra os dados em arquivo, na flecha 6. O processo representado nas flechas 2, 3, 4, 5 e 6 da Figura 4.16 se repete até que todos os eventos de todas as fontes de informação sejam integrados no arquivo de dados final.

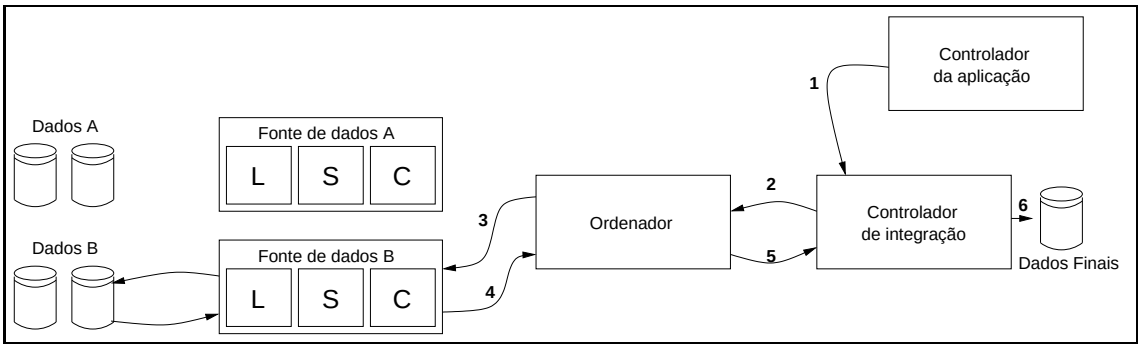

Figura 4.16: Funcionamento geral do protótipo construído e responsável pela integração dos dados coletados por diferentes ferramentas

4.2.2 Sincronização das datas registradas

A sincronização das datas dos eventos registradas pelas ferramentas de coleta é essencial para manter a relação causal entre esses eventos (capítulos 2 e 3). Em alguns casos, não fazer a sincronização dos dados torna impraticável a análise conjunta das informações, pois diferentes fontes de informação podem usar diferentes datas relativas para registrar seus eventos. Para realizar a sincronização de eventos no protótipo implementou-se o método proposto por MAILLET; TRON (1995). Esse método de sincronização consiste em inicialmente registrar as diferenças de relógio entre as máquinas monitoradas em relação a uma máquina de referência. Depois desse processo, utiliza-se essas diferenças para realizar a sincronização dos eventos em relação a máquina de referência escolhida.

O registro das diferenças de relógio deve ser feito antes e depois do período de monitoramento. A aplicação da libRastro `rastro_timesync` foi utilizada para realizar este processo. O resultado da execução dessa aplicação é um arquivo que contém em cada linha o nome e o relógio da máquina de referência seguido pelo nome e relógio de uma das máquinas monitoradas. A Tabela 4.6 mostra um arquivo com informações de sincronização que contém a `paple` como máquina de referência e as máquinas `paple01`, `paple02`, `paple03` e `paple04` como máquinas monitoradas.

Tabela 4.6: Conteúdo de um arquivo contendo o registro das diferenças de relógio entre uma máquina de referência e as máquinas monitoradas

Antes do período de monitoramento				
1	paple	1094221332965040	paple01	1094222792354177
2	paple	1094221333154618	paple02	1094222263029626
3	paple	1094221333343677	paple03	1094222084364163
4	paple	1094221333497292	paple04	1094221463236745
Depois do período de monitoramento				
5	paple	1094221337489491	paple01	1094222796878767
6	paple	1094221337621866	paple02	1094222267496840
7	paple	1094221337752345	paple03	1094222088772874
8	paple	1094221337966650	paple04	1094221467706105

As informações de sincronização semelhantes às mostradas na Tabela 4.6 são enviadas a cada uma das fontes de informações utilizadas no protótipo (Figura 4.15). A sin-

cronização é feita então tomando-se por base essas informações e o local de registro de determinado evento. Por exemplo, se um evento foi registrado na `paple03`, as diferenças de relógio listadas nas linhas 3 e 7 da tabela serão utilizadas para realizar a sincronização. A fórmula 4.1 mostra como é feita a sincronização da data de um determinado evento. Nesta fórmula, t_s é o tempo que se deseja sincronizar e t_s^{ref} é esse mesmo tempo sincronizado de acordo com o relógio da máquina de referência utilizada. Ainda na fórmula, $\left(t_n^{ref} - t_o^{ref}\right)$ representa a diferença entre o tempo da máquina de referência registrado no fim e no início do período de monitoramento, e $(t_n - t_o)$ a mesma diferença mas para os tempos registrados na máquina que tem o tempo que deve ser sincronizado.

$$t_s^{ref} = t_o^{ref} + (t_s - t_o) * \left(\frac{t_n^{ref} - t_o^{ref}}{t_n - t_o}\right) \tag{4.1}$$

Por exemplo, para sincronizar o tempo $t_s = 1094222084364200$ da máquina `paple03` deve-se utilizar os tempos $t_n^{ref} = 1094221337752345$ e $t_o^{ref} = 1094221333343677$ da máquina `paple` e os tempos $t_n = 1094222088772874$ e $t_o = 1094222084364163$ da máquina `paple03`, retirados da Tabela 4.6. Aplicando esses valores na fórmula 4.1, teremos o tempo t_s sincronizado para o tempo $t_s^{ref} = 1094221333343713$.

A aplicação do método de sincronização descrito acima é feita no componente fonte de dados do protótipo. Os tempos dos eventos que são lidos por uma determinada fonte de dados são sincronizados com informações semelhantes as da Tabela 4.6. Cada componente fonte de informação realiza a sincronização dos seus eventos de forma independente em relação aos outros componentes. Os eventos, já sincronizados, são enviados ao integrador de dados após o processo de sincronização e então ordenados de acordo com as datas já sincronizadas.

4.2.3 Unificação da hierarquia dos dados

A unificação da hierarquia dos dados consiste em unir as entidades semelhantes das hierarquias das diferentes fontes de informação. Essa unificação deve ser realizada também nos identificadores utilizados para as instâncias de cada entidade dessa hierarquia. Nesta seção será descrito o processo de unificação hierárquica das fontes de informação utilizadas e a forma como isso foi implementado no protótipo. Esse processo de unificação define também quais são os identificadores utilizados nas entidades da hierarquia.

As fontes de informação utilizadas neste trabalho e descritas na seção anterior foram combinadas de três formas: uma para aplicações desenvolvidas com DECK, outro para aplicações desenvolvidas em MPI e uma terceira na qual somente as ferramentas de monitoramento de *cluster* Ganglia e Performance Co-Pilot são utilizadas. Essas escolhas foram realizadas de forma a mostrar a adaptabilidade do protótipo a combinação de diferentes fontes de informação. A seguir serão explicadas mais detalhadamente como foi feita a unificação das informações nessas três combinações. Ao final da seção será detalhado como foi implementada a unificação das informações.

Unificação de dados para depuração de aplicações DECK

A integração de dados para a depuração de aplicações DECK consiste em unir informações de rastreamento de uma aplicação DECK, dados do sistema operacional e informações coletadas pelas ferramentas de monitoramento de *cluster* Ganglia e Performance Co-Pilot. As Figuras 4.3, 4.13, 4.9 e 4.11, apresentadas na seção 4.1, mostram a hierarquia de entidades dessas fontes de informação, respectivamente.

Para realizar a unificação hierárquica deve-se primeiro identificar as entidades semelhantes e agrupá-las de acordo com essa identificação. A Figura 4.17 mostra a unificação hierárquica entre as fontes de informação utilizadas na combinação. As entidades semelhantes identificadas são: “Máquina”, nas quatro fontes de informação; e “Fluxo de execução” e “Processo” nas fontes de informação DECK e do núcleo do sistema operacional, respectivamente. A entidade “Cluster”, da fonte de informação Ganglia, não foi unificada hierarquicamente com outras entidades pois as outras fontes de informação não contém uma entidade semelhante.

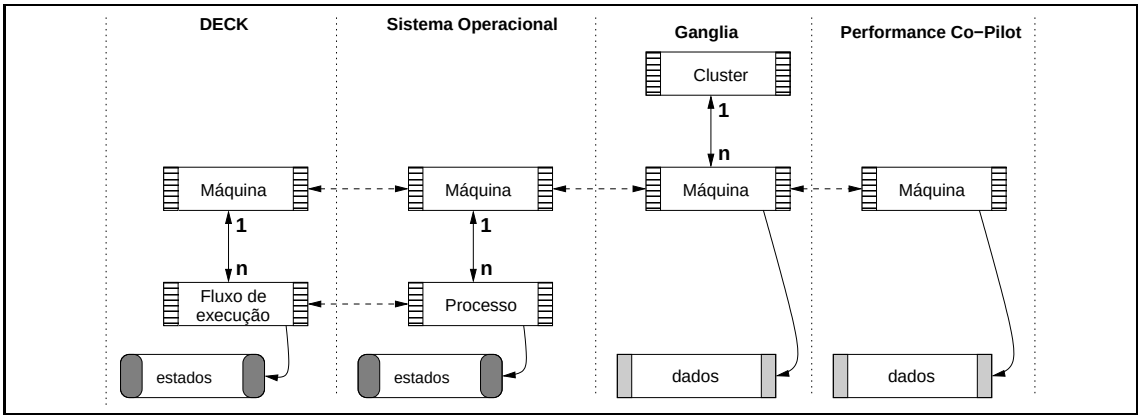

Figura 4.17: Hierarquia de dados das fontes de informação DECK, Sistema operacional Linux, Ganglia e Performance Co-Pilot e identificação das entidades semelhantes

Os identificadores das entidades são os nomes dos nós do *cluster* para a entidade “Máquina” e o número do processo do sistema operacional para as entidades “Fluxo de execução” e “Processo”. As quatro fontes de informação utilizadas nesta combinação utilizam como identificadores para os nós os nomes das máquinas. Dessa forma, nenhuma adaptação teve de ser feita para que a entidade “Máquina” fosse unida. O mesmo acontece para a identificação dos fluxos de execução e processos. As fontes de informação DECK e sistema operacional utilizam os números de processo para identificar essas entidades.

A Figura 4.18 mostra a hierarquia de entidades resultante da unificação das hierarquias das fontes de informação da combinação desta seção. Os dados coletados pelas ferramentas Ganglia e PCP ficam unificadas de acordo com a máquina onde foram coletadas. Os múltiplos fluxos de execução, também agrupados de acordo com a máquina onde foram rastreados, possuem os estados DECK e Sistema operacional Linux (SO).

A unificação hierárquica mostrada na Figura 4.18 é utilizada para aplicações DECK que são executadas em um *cluster*. Caso essa aplicação seja executada em um *Grid*, ou em ambientes Multi-*cluster*, deve-se alterar a hierarquia para conter a entidade “Cluster” do Ganglia. Caso isso ocorra, deve-se criar identificadores para as fontes de informação DECK, Sistema operacional Linux e Performance Co-Pilot. Esses identificadores devem ser então unificados com os identificadores de *cluster* usados pelo Ganglia.

Integração para depuração de aplicações MPI

A outra combinação de fontes de informação consiste em unir dados de rastrea-

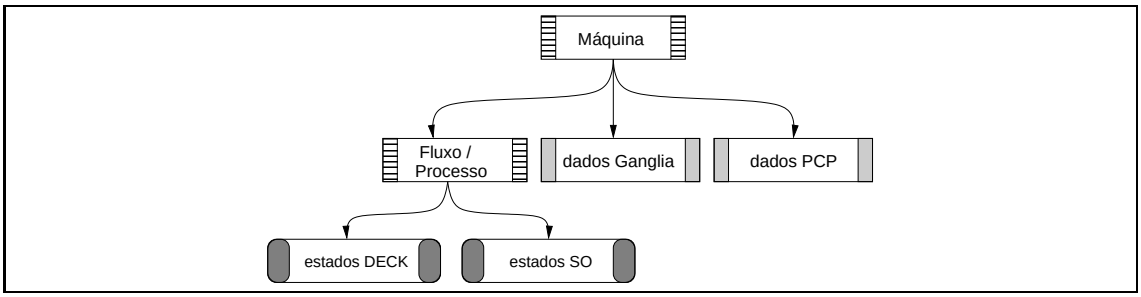

Figura 4.18: Unificação hierárquica das fontes de informação DECK, Sistema operacional Linux, Ganglia e Performance Co-Pilot

mento de uma aplicação MPI, dados do sistema operacional e informações coletadas pelo Ganglia e pelo Performance Co-Pilot. As Figuras 4.6, 4.13, 4.9 e 4.11, descritas na seção 4.1, mostram as entidades dessas fontes de dados, respectivamente. Alguns resultados preliminares do processo de integração de dados de aplicações MPI com Ganglia foram publicados em NEVES; SCHEID; SCHNORR; CHARAO (2004).

A Figura 4.19 mostra as entidades semelhantes das fontes de informação utilizadas na combinação de dados desta seção. As entidades semelhantes, identificadas pelas setas pontilhadas, são a entidade "Máquina" do Sistema operacional Linux, do Ganglia e do PCP e a entidade "Processo", semelhante na fonte de informação do MPI e do sistema operacional. A entidade "Cluster" não foi unificada, pois toma-se como base a execução de aplicações MPI em apenas um *cluster*. Caso isso não aconteça, deve-se realizar a unificação dessa entidade, como descrito na seção anterior.

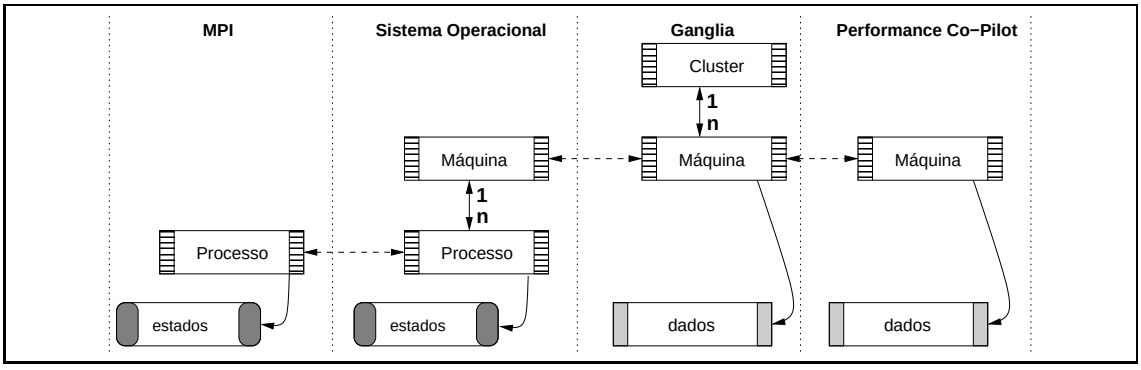

Figura 4.19: Identificação das entidades semelhantes das fontes de informação MPI, Sistema operacional Linux, Ganglia e Performance Co-Pilot

A Figura 4.20 mostra a unificação hierárquica das fontes de informação MPI, SO, Ganglia e PCP. Nessa unificação, as entidades "Máquinas" das fontes de informação SO, Ganglia e PCP foram unificadas. A entidade "Processo" das fontes de dados SO e MPI também foram unificadas. Como pode ser visto na Figura 4.19, a entidade "Máquina" da hierarquia MPI não existe. Para que a unificação representada na Figura 4.20 possa ser viabilizada teve-se que mapear os processos MPI para o local onde esses processos eram executados. Esse mapeamento é feito utilizando-se o arquivo que define em quais nós do *cluster* será executada a aplicação MPI. O

ambiente de execução MPI distribui os processos de acordo com esse arquivo. Essa distribuição coloca os processos na ordem em que as máquinas estão no arquivo. Com esse arquivo e essa forma de distribuição de processos do MPI, é possível criar durante a integração a entidade “Máquina” para essa fonte de informação e então unificar os dados como mostrado na Figura 4.20.

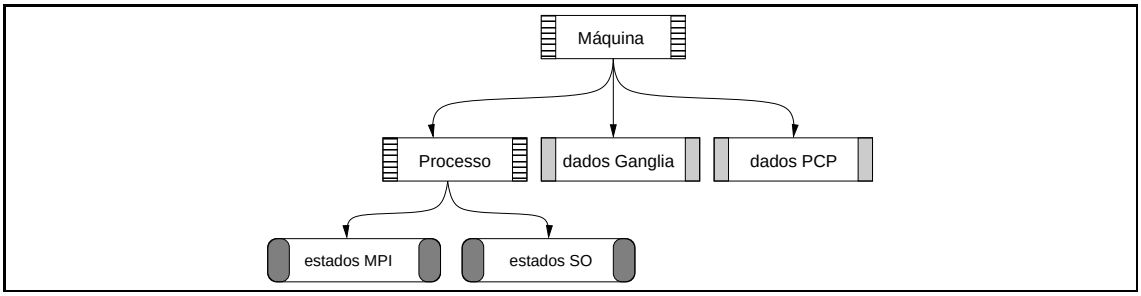

Figura 4.20: Unificação hierárquica das fontes de informação MPI, Sistema Operacional Linux, Ganglia e PCP

Para permitir a unificação hierárquica, deve-se utilizar os mesmos identificadores para as entidades semelhantes da hierarquia. Na combinação de dados do MPI, SO, Ganglia e PCP, as entidades “Máquina” e “Processo” devem ter seus identificadores unificados. A fonte de informação do Sistema operacional Linux utiliza os números dos processos para identificá-los. No MPI, a identificação de processos é feita através de números seqüenciais. Durante o processo de rastreamento, visto na seção 4.1.2, foi apresentada uma solução que facilita a integração dos dados de processos MPI com os dados de processos do sistema operacional. Essa solução inclui no rastro de execução das aplicações MPI o número de processo do sistema operacional para cada processo MPI. Com essa informação, é possível utilizar o mesmo identificador para as entidades “Processo” do MPI e SO. Os identificadores da entidade “Máquina” são os nomes dos nós. Na fonte de dados MPI, esse identificador é obtido através do mapeamento dos processos MPI para onde eles são executados, como foi descrito anteriormente. Nas outras fontes de dados utilizadas, o identificador é obtido diretamente dos arquivos gerados na etapa de coleta.

Integração para monitoramento de *cluster*

A última combinação utilizada consiste em unir dados coletados pelas ferramentas de monitoramento Ganglia e Performance Co-Pilot. As Figuras 4.9 e 4.11, descritas na seção 4.1, mostram as hierarquias de dados dessas ferramentas respectivamente. Com essa combinação de fontes de informação, é possível incrementar a análise dos dados através da visualização de informações complementares de monitoramento registradas por mais de uma ferramenta de coleta.

A Figura 4.21 mostra a identificação das entidades semelhantes das ferramentas Ganglia e PCP. A entidade “Máquina” é a única entidade semelhante a ser unida. A entidade “Cluster” da ferramenta Ganglia não tem equivalente no PCP. Caso fosse necessário usar a ferramenta PCP e Ganglia em ambientes como *Grid* ou Multi-*cluster*, deveria ser criada uma forma de identificar um *cluster* nesta hierarquia. Esse processo é parecido com aquele feito na criação da entidade “Máquina” da hierarquia da fonte de informação MPI.

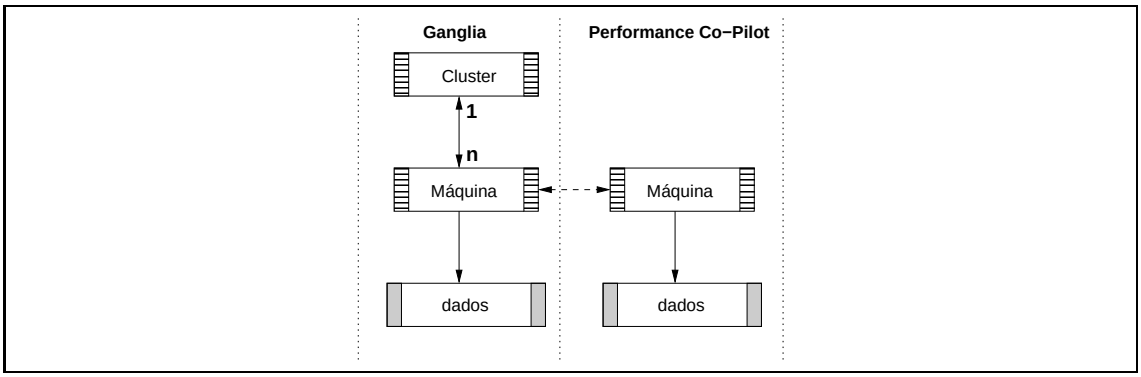

Figura 4.21: Identificação das entidades semelhantes das ferramentas de monitoramento Ganglia e Performance Co-Pilot

A Figura 4.22 mostra a unificação hierárquica das duas ferramentas de monitoramento, Ganglia e Performance Co-Pilot, com a entidade "Máquina" unificada. Os identificadores utilizados para essa entidade nas duas ferramentas de monitoramento é o nome do máquina, de forma que não é necessário nenhum tipo de conversão para unificar esses identificadores.

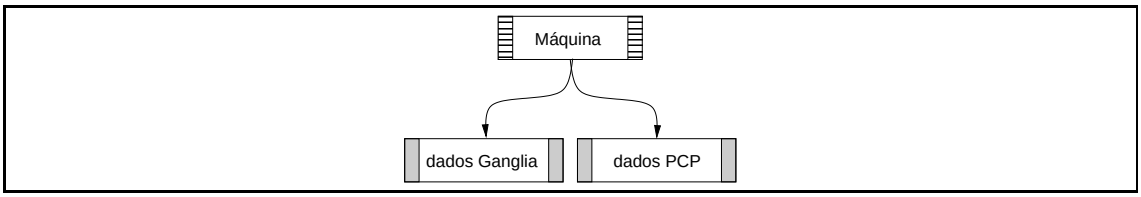

Figura 4.22: Unificação hierárquica dos dados das ferramentas de monitoramento Ganglia e Performance Co-Pilot

Funcionamento no protótipo

A unificação hierárquica é implementada no protótipo através da utilização de arquivos de configuração utilizados como parâmetro para o controlador de integração. Cada um desses arquivos descreve a hierarquia de dados unificada para uma combinação específica de fontes de informação. Por exemplo, existe um arquivo com a descrição da unificação para ferramentas de monitoramento Ganglia e Performance Co-Pilot, outro para a depuração de aplicações DECK. As informações de cada um dos arquivos são utilizados também para gerar o cabeçalho do arquivo de saída no formato Pajé e para, juntamente com informações dos componentes fonte de dados, construir identificadores para as instâncias das entidades da hierarquia.

Foi criado um arquivo de configuração com a descrição da unificação hierárquica para cada uma das combinações de fontes de informação explicadas anteriormente, ou seja, integração de dados para a depuração de aplicações DECK, MPI e integração de dados das ferramentas de monitoramento Ganglia e PCP. Esses arquivos são utilizados de acordo com as informações que se deseja integrar. Por exemplo, caso se esteja interessado em integrar informações oriundas do rastreamento de uma aplicação MPI com dados de monitoramento do Ganglia, basta utilizar o arquivo com a configuração da unificação hierárquica correspondente.

A Figura 4.23 ilustra o funcionamento da criação de identificadores realizada pelo protótipo. Na inicialização do protótipo é passado como parâmetro o arquivo de configuração que descreve a unificação de dados que será realizada. Após isso, quando alguma fonte de dados utilizada necessita de um identificador, ela envia uma mensagem para o controlador de integração contendo os dados necessários para a construção desse identificador. O controlador de integração, baseado nas informações do arquivo de unificação hierárquica e nos dados passados por determinada fonte de dados, constrói o identificador e o envia de volta ao componente que requisitou o identificador. Essa forma de funcionamento tem como maior vantagem a centralização da geração de identificadores.

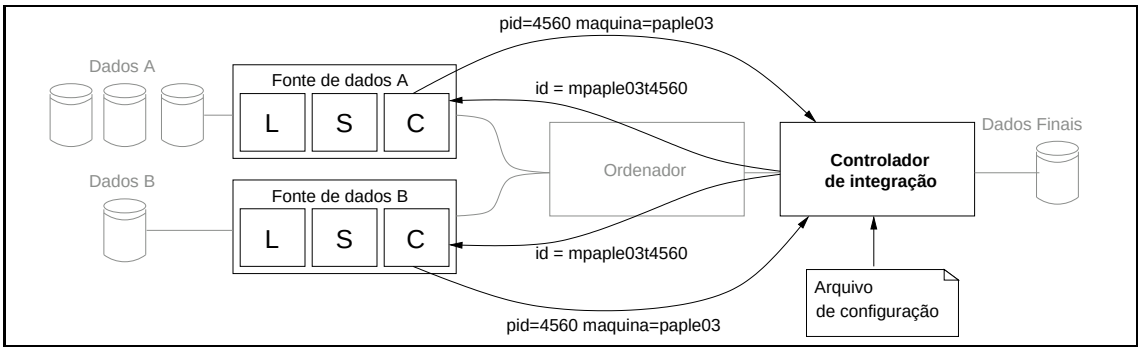

Figura 4.23: Forma de funcionamento da criação de identificadores para as instâncias das entidades das fonte de dados

4.2.4 Padronização de formato

Até o momento foram vistas a forma de funcionamento da sincronização e da unificação hierárquica e de identificadores. A padronização dos dados ocorre juntamente com essas outras duas partes e consiste basicamente em transformar o formato das informações para o formato da ferramenta de visualização Pajé. Essa transformação se caracteriza por ser feita através do mapeamento de um determinado evento da fonte de dados em um ou um conjunto de eventos Pajé correspondentes, de acordo com as capacidades da ferramenta de visualização, descrita no apêndice A.5, e como se deseja visualizar os dados da fonte de dados. Esse mapeamento é feito na implementação do protótipo através da conversão dos eventos, no componente fonte de dados. Dessa forma, cada componente fonte de dados tem um conversor que sabe padronizar todos os eventos que possam ser lidos por essa fonte de dados.

O funcionamento do processo de padronização funciona no conversor (C) do componente fonte de dados. Os eventos que são lidos pelo leitor (L) e sincronizados (S) são convertidos de acordo com o tipo desses eventos. O conjunto de eventos resultante desse processo de conversão é enviado para o ordenador de dados e por fim ao controlador de integração, que realiza a escrita desses eventos no arquivo de saída.

Ao longo desta seção serão descritos os mapeamentos realizados para os eventos gerados pelas fontes de informação DECK, MPI, Ganglia, sistema operacional Linux e Performance Co-Pilot. Essa descrição consiste em mostrar a forma como se quer visualizar as informações de cada uma dessas fontes e como foi feito o mapeamento dos eventos para o formato Pajé.

DECK

Na seção 4.1.1 foram vistos os pontos de instrumentação detectados e utilizados no rastreamento de uma aplicação DECK. Esses pontos de instrumentação se localizam nas funções de comunicação e gerência de fluxos de execução da biblioteca. A Figura 4.24 mostra a visualização desejada do comportamento de uma aplicação DECK quando a biblioteca é instrumentada. Essa visualização esquemática é baseada em quais eventos foram registrados durante o período de rastreamento de uma aplicação DECK e nas capacidades da ferramenta de visualização Pajé. Na Figura 4.24, os fluxos de execução são representados por retângulos horizontais e os diferentes estados que esse pode obter, por cores diferentes nesses retângulos. As flechas indicam comunicações entre os diferentes processos de uma aplicação DECK. Com essa visualização, é possível identificar eventos que acontecem em um determinado fluxo de execução de forma independente, e também a interação entre esses fluxos e o comportamento resultante dessas interações.

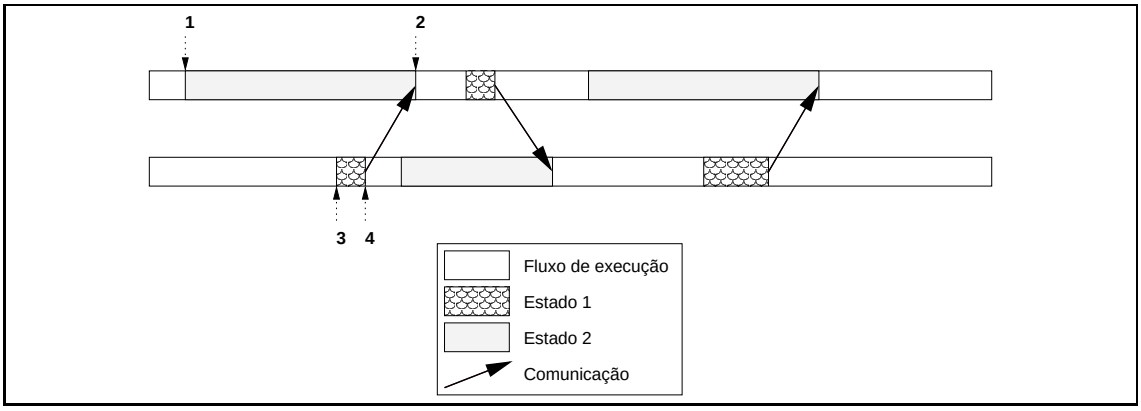

Figura 4.24: Visualização esquemática do comportamento de uma aplicação desenvolvida com o DECK

A Figura 4.25 mostra um exemplo de como é feito o mapeamento dos eventos DECK para os eventos Pajé para se atingir a visualização apresentada na Figura 4.24. Neste exemplo, durante a comunicação entre dois processos de uma aplicação DECK, o processo que envia o dado executa a função `mbox_post` e o processo que recebe o dado executa a função `mbox_retrv`. Assim, quatro eventos são registrados: `mbox_post_in` (tempo 2), e `mbox_post_out` (tempo 3), no processo que enviou o dados; e os eventos `mbox_retrv_in` (tempo 1) e `mbox_retrv_out` (tempo 4) no processo que recebeu os dados. O mapeamento consiste então em mapear cada um desses eventos para os eventos Pajé capazes de obter a visualização desejada. Esse mapeamento pode ser visto na parte inferior da Figura 4.25. No exemplo, o evento `mbox_post_in` foi mapeado para o evento `PajeSetState(post)`, que é responsável por trocar a cor do retângulo horizontal. O evento `mbox_post_out` foi mapeado para os eventos `PajeSetState(exec)` e `PajeStartLink(com)`, sendo este evento o responsável por iniciar uma flecha. O evento `mbox_retrv_in` foi mapeado para `PajeSetState(recv)` e o evento `mbox_retrv_out` foi mapeado para os eventos `PajeSetState(exec)` e `PajeEndLink(com)`, sendo este o evento que representa o fim de uma flecha.

O mapeamento dos outros eventos registrados no período de rastreamento de uma aplicação DECK (Tabela 4.1) acontece de forma semelhante ao exemplo dado anteriormente. No apêndice C.1 estão listados todos os eventos gerados pela biblioteca DECK e os eventos Pajé correspondentes para se atingir a visualização desejada.

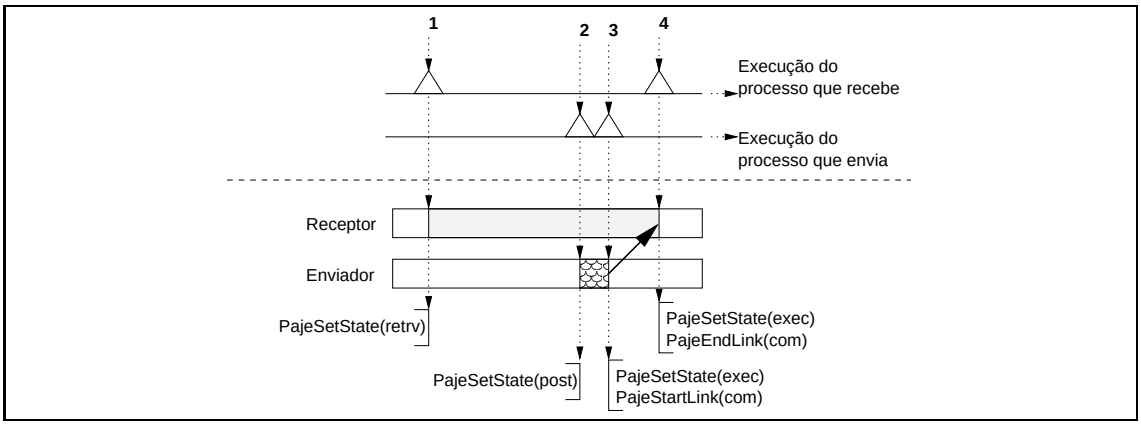

Figura 4.25: Funcionamento do mapeamento dos eventos gerados durante a execução de uma aplicação DECK para eventos Pajé

MPI

Na seção 4.1.2 foi descrita a biblioteca MPE que realiza o rastreamento das comunicações de uma aplicação MPI. Esse rastreamento tem como característica a geração de três estados diferentes que cada processo da aplicação pode estar: recebendo, enviando e executando. A visualização desejada de uma aplicação MPI com base nos eventos registrados na etapa de rastreamento é semelhante aquela mostrada anteriormente para aplicações DECK. Ela consiste em um conjunto de retângulos horizontais para representar os diferentes processos e três cores diferentes para representar cada estado desses processos. A Figura 4.26 mostra essa visualização de forma esquemática. Esse formato de visualização, assim como na do DECK, permite a identificação dos estados de cada processo e a interação entre eles.

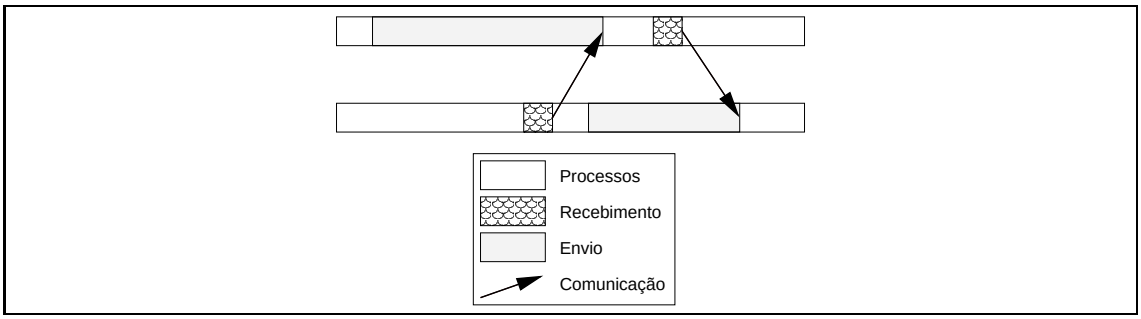

Figura 4.26: Visualização esquemática desejada do comportamento de uma aplicação desenvolvida com MPI

O mapeamento dos eventos gerados pela MPE para os eventos Pajé correspondentes foi feito de forma semelhante ao exemplo apresentado anteriormente, na seção de padronização de eventos DECK. Esse mapeamento de eventos procurou se aproximar da visualização desejada para os dados de rastreamento MPI. Por exemplo, o evento `mpi_send_in`, que indica a entrada de um processo no estado de envio, foi mapeado para o evento Pajé `PajeSetState(send)`, que altera a cor da barra horizontal que representa um processo. O evento `mpi_send_out`, que indica o fim do estado de envio, foi mapeado para outro evento `PajeSetState(exec)`, alterando a cor da visualização para aquela que representa

o estado de execução. Entre esses dois eventos da biblioteca MPE é registrado o evento `mpi_send`, que indica o envio de dados do processo para outro naquele instante. Esse evento foi mapeado para o evento `PajeStartLink`, que indica o início de uma flecha na visualização.

O mapeamento dos outros eventos registrados pela biblioteca MPE para aplicação MPI foram feitos de forma semelhante ao exemplo do parágrafo anterior. A tabela com a lista desses mapeamentos está no apêndice C.2.

Ganglia

O rastreamento das informações coletadas pelo Ganglia foi descrito na seção 4.1.3. Esse rastreamento consiste em capturar informações sobre a utilização dos recursos de uma determinada máquina do *cluster*. Essas informações se caracterizam por serem eventos quantitativos em um determinado instante de tempo. Por exemplo, em determinado instante a utilização da CPU era de 96%, ou a quantidade de *bytes* por segundo chegando em uma placa de rede era de 30Kbits. Com esses dados e as potencialidades da ferramenta Pajé, é possível construir um gráfico para cada métrica coletada pelo Ganglia. A Figura 4.27 mostra uma possível visualização gráfica de dois dados coletados pelo Ganglia. Cada gráfico representa a evolução de determinada métrica ao longo do tempo. Com essa forma de visualização é possível identificar alterações nos valores das métricas sem precisar recorrer a tabelas ou informações textuais.

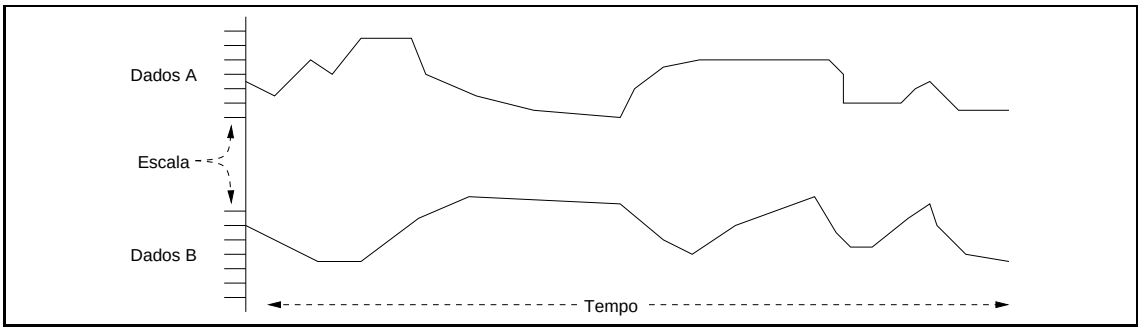

Figura 4.27: Visualização esquemática dos dados coletados pelo rastreamento do Ganglia

Pajé tem um conjunto de eventos específico para visualizar graficamente dados variáveis ao longo do tempo. O mapeamento dos eventos Ganglia foi feito utilizando-se esse conjunto de eventos Pajé. Por exemplo, quando um evento do Ganglia define a quantidade de dados trafegados na rede em um determinado instante, utiliza-se o evento Pajé do tipo `PajeSetVariable` para dizer o valor de determinada entidade em um instante de tempo. Realizando várias vezes a conversão dos eventos de quantização para esse evento Pajé resulta em uma visualização gráfica semelhante a vista na Figura 4.27. Os mapeamentos que foram feitos dos eventos do Ganglia são descritos no apêndice C.3.

Sistema Operacional Linux

O rastreamento das trocas de contexto do sistema operacional Linux foi descrito na seção 4.1.5. As informações coletadas por esse rastreamento são eventos que dizem em qual instante determinado processo foi posto em execução ou bloqueado. Com essas informações e as características da ferramenta Pajé é possível construir uma visualização semelhante aquela mostrada nas Figuras 4.24 e 4.4, onde os retângulos horizontais repre-

sentam determinado processo e os estados são representados por diferentes cores nesses retângulos.

O mapeamento utilizado para se atingir a visualização desejada é semelhante aos utilizados nas informações coletadas em uma aplicação DECK e em uma aplicação MPI. Por exemplo, o evento de troca de contexto no qual o processo vai ser posto em execução em um determinado instante é mapeado para o evento Pajé que troca a cor do retângulo horizontal. O mesmo ocorre quando um evento registrado diz que determinado processo é bloqueado. O apêndice C.4 especifica os mapeamentos feitos na conversão dos eventos registrados no Sistema operacional Linux para os eventos Pajé.

Performance Co-Pilot

As informações coletadas pelo Performance Co-Pilot dizem respeito a utilização de recursos de uma determinada máquina do *cluster*. Estas informações normalmente são dados quantitativos que podem ser melhor representados através de gráficos.

O PCP se caracteriza por ter somente um tipo de evento. Nesses eventos, tem-se basicamente dois campos: descrição e valor. O campo descrição diz a que se refere o valor e onde esse valor deve entrar dentro da hierarquia de informações da ferramenta. O campo valor tem a informação útil do evento.

O mapeamento dos eventos do PCP para eventos Pajé acontece de forma semelhante ao mapeamento utilizado para os eventos coletados pelo Ganglia. Esse mapeamento consiste em utilizar o evento `PajeSetVariable` para gerar gráficos na visualização. A Figura 4.28 mostra um mapeamento exemplo de dados de utilização de CPU (all.cpu.load) coletados pelo PCP e visualizados esquematicamente através de um gráfico. O apêndice C.5 mostra os mapeamentos feitos na conversão dos eventos registrados pelo Performance Co-Pilto para os eventos Pajé.

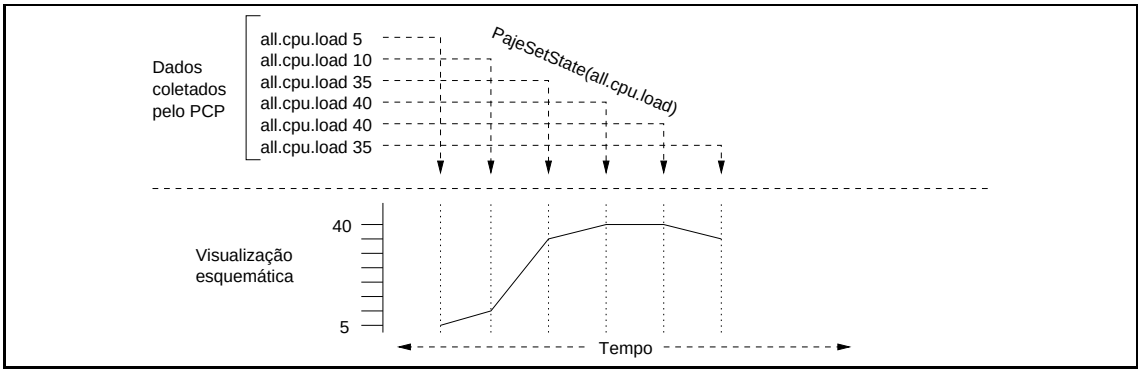

Figura 4.28: Mapeamento dos eventos coletados pelo Performance Co-Pilot para a visualização no Pajé

4.2.5 Diagrama de classes

O protótipo foi construído utilizando a linguagem orientada a objetos Objective-C. A orientação a objeto foi escolhida por ter características como fácil manutenção e ampliação do programa, caso necessário. Nesta seção serão apresentados os diagramas de classes utilizados no protótipo. Serão mostradas as principais classes do protótipo e as classes auxiliares.

A Figura 4.29 mostra o diagrama de classes do protótipo. A classe `FileReader` e

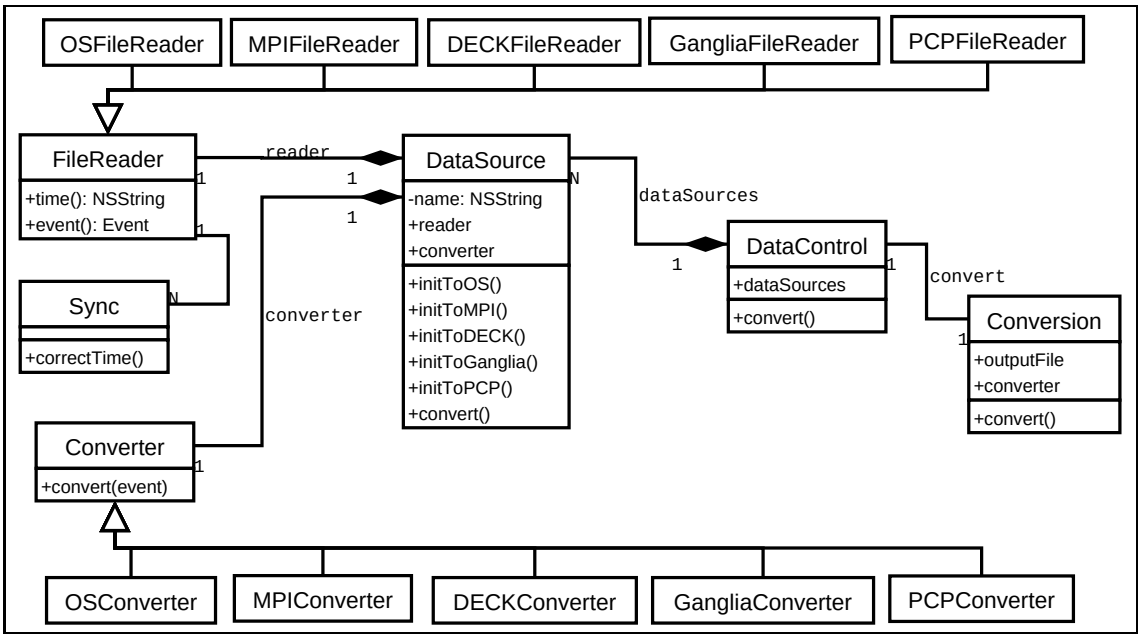

Figura 4.29: Diagrama das principais classes do protótipo construído

`Converter` são classes abstratas que contém métodos e atributos que são utilizados pela classe `DataSource` e que devem ser implementados pelas sub-classes. Foram construídos cinco sub-classes da classe `FileReader` e cinco da classe `Converter`. Essas sub-classes foram construídas de acordo com a forma de leitura do formato de arquivo, no caso das sub-classes de `FileReader`, e na forma de padronização dos dados através da conversão, no caso das sub-classes de `Converter`.

A classe `DataSource` na Figura 4.29 é a implementação do componente fonte de informação, descrito na seção 4.2 deste capítulo. Essa classe contém um ou mais objetos de uma determinado sub-classe de `FileReader` e um objeto do mesmo tipo da sub-classe de `Converter`. Por exemplo, caso `DataSource` seja inicializado para ser uma fonte de informação DECK, ela instanciará um `DECKFileReader` para cada arquivo que deve ser integrado e um objeto da classe `DECKConverter`. Ao receber uma mensagem para executar o método `convert`, um objeto da classe `DataSource` lê o evento através do objeto `reader` e realiza a conversão do evento através do objeto `converter`. O resultado dessa operação é passado a quem chamou o método `convert`. Cada classe `DataSource` tem um ou mais objetos da classe `Sync`, responsáveis por realizar a sincronização dos eventos.

A classe `DataControl` na Figura 4.29 é a implementação do componente ordenador de dados do protótipo. Essa classe é inicializada com uma descrição das informações que devem ser integradas e instancia tantos objetos da classe `DataSource` quantos forem necessários de acordo com essa descrição.

A classe `Conversion` na Figura 4.29 é a implementação do controlador de integração. Ele mantém uma referência para a classe `DataControl`. A classe `Conversion`, depois de inicializada, chama o método `convert` da classe `DataControl`. A classe `DataControl` decide quais dos objetos `DataSource` tem o evento com a data mais recente e chama o método `convert` dessa classe.

A Figura 4.30 mostra as classes implementadas para descrever os eventos das diferentes fontes de informação utilizadas no protótipo. Essa descrição é utilizada internamente no protótipo. As sub-classes de `FileReader` retornam objetos das sub-classes de `Event`

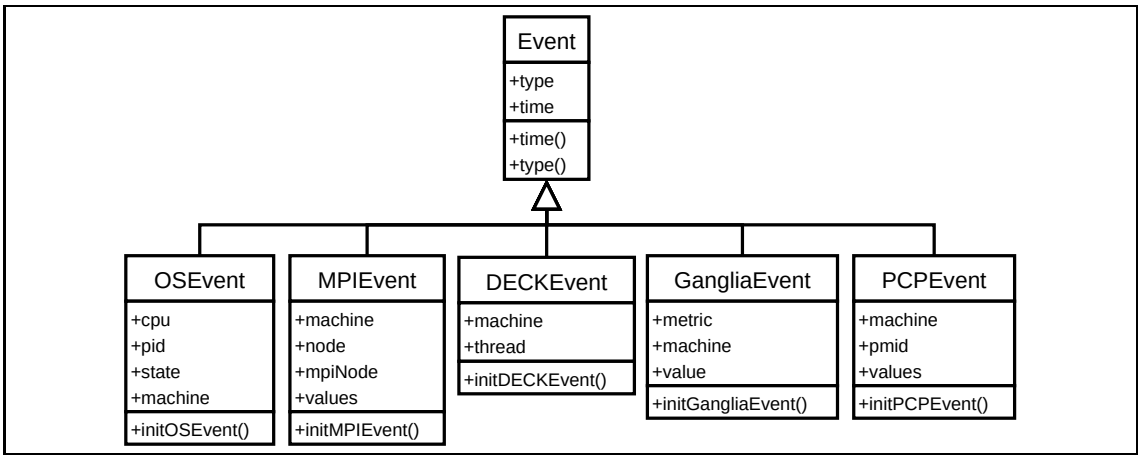

Figura 4.30: Diagrama de classes dos eventos possíveis implementados no protótipo

de acordo com a fonte de dados do evento. Esses objetos são então tratados pelas respectivas sub-classes de `Converter`. Por exemplo, a classe `PCPFileReader` retorna objetos da classe `PCPEvent` que são tratados pela classe `PCPConverter`.

4.3 Visualização Integrada

Na seção 4.2.3 foram vistas três formas de unificação hierárquica das fontes de informação implementadas no protótipo. Essas três formas são as seguintes: combinação de dados de rastreamento de aplicações MPI com dados do sistema de monitoramento de *cluster*, dados de aplicações DECK com dados de monitoramento e somente dados de monitoramento com duas ferramentas de monitoramento. Depois, na seção 4.2.4, foram mostradas a padronização e a forma de visualização desejada para cada fonte de dados do protótipo.

Na seção atual serão mostradas as formas desejadas de visualização integradas. Inicialmente, serão mostradas a integração de dados para a depuração de aplicações DECK e MPI. Esses dados são oriundos do rastreamento das aplicações em si, do sistema operacional e das ferramentas de monitoramento. No final, será mostrado somente a integração de informações das ferramentas de monitoramento Ganglia e Performance Co-Pilot.

4.3.1 Depuração de aplicações

A seção 4.2.4 apresentou as formas de visualização esquemáticas de dados rastreados em aplicações DECK e MPI, sistema operacional Linux e ferramentas de monitoramento Ganglia e Performance Co-Pilot. Nesta seção será mostrada a forma integrada de visualização dessas informações.

A Figura 4.31 mostra a forma desejada de visualização integrada das informações dessas fontes de dados. Nessa figura, é apresentada uma visualização hipotética de um programa executado em dois nós sendo que no nó 0 são executados dois processos. O monitoramento do Ganglia obtém dados de utilização de CPU e o monitoramento do Performance Co-Pilot mostra dados da quantidade de memória livre. As informações de cada nó coletadas por diferentes fontes são agrupadas. Os estados que o rastreamento de um determinado processo da aplicação paralela são mostrados paralelamente aos estados obtidos pelo mesmo processo dentro do sistema operacional. As informações das

ferramentas de monitoramento de *cluster* são mostradas abaixo dos estados obtidos pela aplicação paralela.

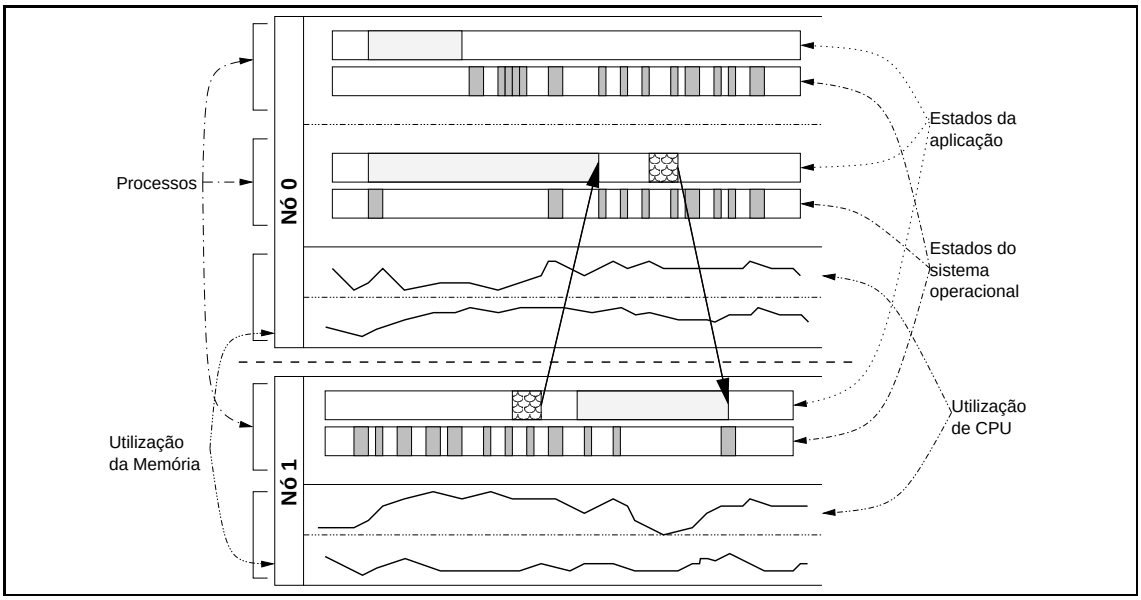

Figura 4.31: Forma de visualização integrada que se deseja obter das informações registradas por aplicações DECK ou MPI, sistema operacional e ferramentas de monitoramento Ganglia e Performance Co-Pilot

Essa visualização da Figura 4.31 permite observar as relações que existem entre os eventos e estados das diferentes fontes de informação em especial entre os dados obtidos através da aplicação e do sistema operacional. É possível também correlacionar visualmente os dados das ferramentas de monitoramento com os estados atingidos pela aplicação. O capítulo 5 apresentará exemplos da depuração de aplicações paralelas desenvolvidas tanto em DECK quanto em MPI com essa integração visual dos dados no Pajé.

4.3.2 Monitoramento de *cluster*

Na seção 4.2.4 foram apresentadas as formas de visualização esquemática de dados do Ganglia e do Performance Co-Pilot. Esta seção mostra a integração visual das informações que se deseja obter com a ferramenta de visualização Pajé.

A Figura 4.32 mostra a forma desejada de visualização integrada das informações das ferramentas de monitoramento Ganglia e Performance Co-Pilot. Nessa figura, tem-se uma situação hipotética na qual as duas ferramentas monitoram dados de dois nós diferentes. No nó 0 as informações `cpu_user`, `bytes_in`, `bytes_out` são coletadas pelo Ganglia e o dado `mem_free` é coletado pelo PCP. Nessa situação, o dado `cpu_user` não está disponível no nó 1. Assim como na depuração de aplicações paralelas, as informações coletadas por diferentes fontes de informação são agrupadas de acordo com o nó onde foram obtidas. Esse agrupamento obedece a unificação hierárquica de entidades definida na seção 4.2.3 deste capítulo, permitindo uma melhor correlação das informações obtidas internamente dentro de um nó e de dados gerados em duas máquinas diferentes. O capítulo 5 apresentará exemplos da visualização no Pajé das informações coletadas pelas ferramentas de monitoramento utilizadas no trabalho.

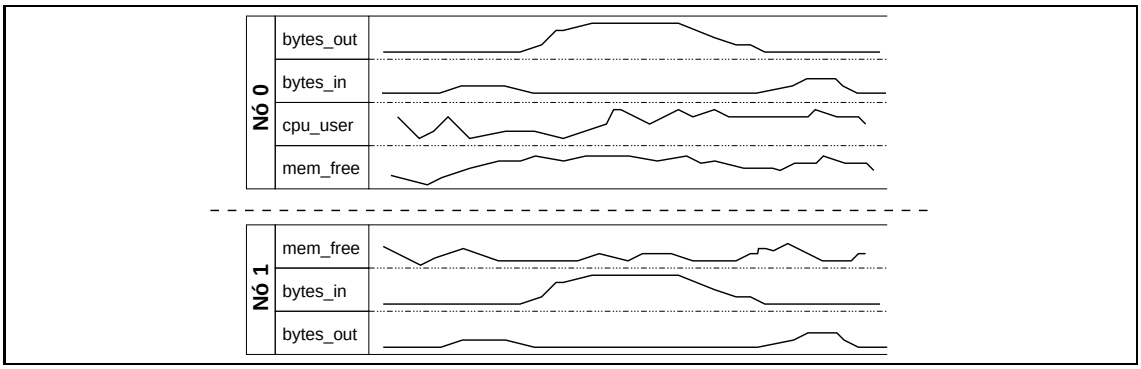

Figura 4.32: Forma de visualização integrada que se deseja obter das informações registradas pelas ferramentas de monitoramento Ganglia e PCP

4.4 Conclusão do capítulo

Neste capítulo foram apresentados os coletores de informações tanto na área de depuração quanto na de monitoramento de *cluster*. Esses coletores funcionam como fontes de dados para o protótipo desenvolvido neste trabalho, descrito posteriormente, e que foi baseado no modelo de integração de dados do capítulo 3.

No próximo capítulo serão apresentados os resultados do processo de integração obtidos com o protótipo desenvolvido. Esses resultados foram feitos de forma a mostrar informações coletadas por diferentes ferramentas de coleta sendo visualizadas de forma conjunta na ferramenta de visualização Pajé.

5 VALIDAÇÃO

No capítulo anterior foram descritas as fontes de dados utilizadas no trabalho e a ferramenta protótipo de integração criada. Essa ferramenta foi criada com o intuito de integrar os dados que são coletados pelas fontes de dados utilizadas no trabalho. Neste capítulo serão apresentados os resultados obtidos com a integração dos dados feita pelo protótipo do capítulo anterior.

A apresentação dos resultados é feita de forma a mostrar informações coletadas por diferentes fontes de dados sendo visualizadas de forma integrada na ferramenta de visualização Pajé. A validação da integração das informações acontece através da verificação dos resultados em termos de sincronização das informações, da unificação hierárquica e de identificadores e da padronização das informações.

Este capítulo está organizado da seguinte forma. Inicialmente, será descrito o ambiente utilizado para a captura das informações de monitoramento de *cluster* e para o rastreamento de aplicações. Depois serão mostrados os resultados da integração dos dados através de figuras mostrando a janela de visualização do Pajé. Essas figuras mostrarão a integração de dados de ferramentas de monitoramento de *cluster* diferentes, de dados de aplicações DECK e no final de dados de aplicações MPI.

5.1 Descrição do *cluster* utilizado

Para a obtenção dos dados que serão apresentados neste capítulo foi utilizado o *cluster* do Laboratório de Sistemas de Computação da UFSM. Esse *cluster* é composto de dez computadores interconectados através de um chaveador Gigabit Ethernet e de um chaveador Fast Ethernet. A máquina `paple` é o *front-end* do *cluster* e é a única que tem acesso externo. Esta máquina está ligada somente a rede de interconexão Fast Ethernet e oferece serviços como o sistema de arquivos e a gerência de usuários. A Figura 5.1 mostra a organização desse *cluster*.

Cada máquina do *cluster* foi preparada com o objetivo de ter as cinco fontes de informação: Ganglia, Perfomance Co-Pilot, biblioteca DECK instrumentada, MPI com a biblioteca de rastreamento MPE e o núcleo do Sistema operacional Linux alterado para realizar o rastreamento de processos.

Ganglia: Apenas o coletor local de dados, `gmond`, foi instalado. A aplicação auxiliar criada para registrar as informações coletadas pelos `gmond`‘s foi instalada apenas no *front-end*. Esse aplicação se conecta a um dos `gmond`‘s das máquinas do *cluster* e é capaz de obter informações das outras máquinas. São coletadas informações de utilização de CPU, memória, disco e comunicação na placa de rede. A filtragem desses dados é feita através da aplicação auxiliar, como visto na seção 4.1.3.

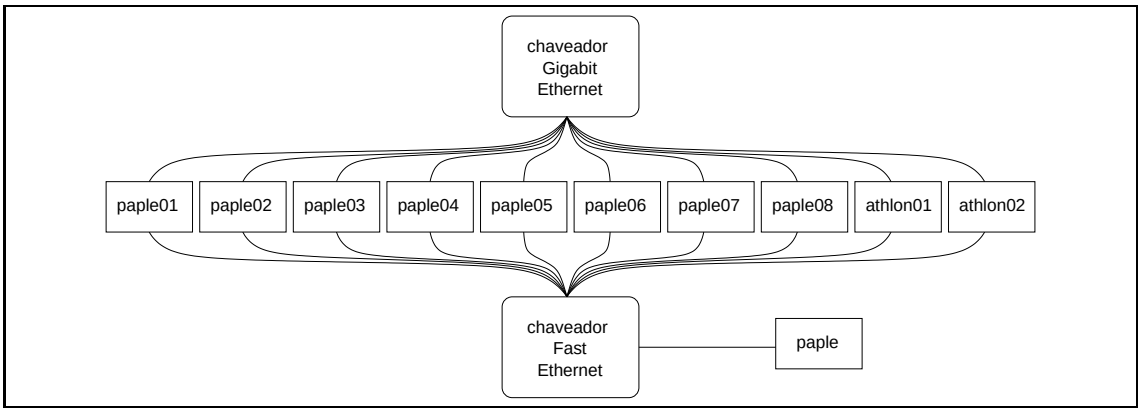

Figura 5.1: Organização da arquitetura do *cluster* do Laboratório de Sistemas de Computação da UFSM

Performance Co-Pilot: Foi feita apenas a instalação padrão da ferramenta. Essa instalação padrão da ferramenta consiste na instalação em cada máquina de um coletor local (`pmcd`) e da aplicação que registra os dados em arquivo, chamada `pmlogger`. Essa aplicação de registro de dados foi configurada de forma a registrar os dados localmente. São coletadas informações da utilização da CPU, memória, disco, sistema de arquivos, interrupções, comunicação na placa de rede.

Núcleo do Linux: Foi colocada em cada máquina do *cluster* uma versão do núcleo do Linux com as implementações descritas na seção 4.1.5. Essa versão é capaz de realizar o rastreamento de processos no núcleo do sistema e é baseada na distribuição original 2.6.2 do núcleo do Linux.

Bibliotecas de rastreamento: As bibliotecas DECK instrumentada e MPE foram instaladas em cada máquina do *cluster*, permitindo a execução de aplicações desenvolvidas e o rastreamento das mesmas.

5.2 Visualização integrada no Pajé

A validação do modelo de integração de dados se dá inicialmente através da coleta de informações pelas diferentes ferramentas, descritas na seção 4.1. Após essa etapa, os dados são integrados através do protótipo de integração descrito na seção 4.2.

A metodologia empregada para realizar a validação do modelo de integração foi classificar a integração de dados em três diferentes tipos: a área de monitoramento de *cluster*, depuração de aplicações desenvolvidas com DECK e depuração de aplicações com a biblioteca MPI. O primeiro tipo, de monitoramento de *cluster*, trata da integração de informações de diferentes ferramentas de *cluster*. O segundo tipo, depuração de aplicações paralelas, consiste na coleta e integração de dados da biblioteca DECK instrumentada, com dados do sistema operacional e informações externas a aplicação como dados de monitoramento de *cluster*. O terceiro tipo, semelhante ao segundo, trata da coleta e integração de dados de informações da biblioteca MPI, do sistema operacional e dados de monitoramento de *cluster*.

5.2.1 Monitoramento integrado de *cluster*

No capítulo 4, seções 4.1.3 e 4.1.4, foram vistas as ferramentas de monitoramento de *cluster* Ganglia e Performance Co-Pilot. Essas duas ferramentas tem um conjunto de dados coletados e intervalo de coleta diferentes. Nesta seção serão mostrados os resultados obtidos com a integração de dados dessas duas ferramentas. Esses resultados estão divididos em seis partes: (1) monitoramento da utilização da rede de um *cluster*, (2) utilização de CPU das máquinas, (3) utilização de memória e o (4) monitoramento de processos de cada máquina que compõem um *cluster*. Depois será apresentada a (5) visualização integrada conjunta de dados de memória, CPU, quantidade de processos e monitoramento da rede. Ao final da seção será apresentada uma (6) visualização integrada de informações coletadas em um ambiente similar a um *Grid*. Todas as informações que serão mostradas nos resultados a seguir foram coletadas ou pelo Ganglia ou pelo Performance Co-Pilot.

Utilização da rede

As ferramentas de monitoramento Ganglia e Performance Co-Pilot são capazes de monitorar valores relacionadas à utilização da rede de um *cluster*. Esses valores são coletados em cada máquina e quando analisados conjuntamente mostram o comportamento como um todo da rede de um *cluster*.

A Figura 5.2 mostra a visualização integrada de informações de dados coletados em três máquinas de um *cluster*: `athlon01`, `paple07` e `paple08`. Os valores `BYTES_OUT`, `PKTS_IN`, `PKTS_OUT` e `BYTES_IN` foram coletadas pela ferramenta Ganglia, enquanto que os dados `sockstat-tcp-inuse`, `sockstat-udp-inuse` e `tcpconn-established` foram coletadas pelo Performance Co-Pilot. Os valores `BYTES_OUT` e `BYTES_IN` indicam a quantidade de dados que está saindo ou chegando a uma determina máquina em um determinado intervalo de tempo. O mesmo para os valores `PKTS_OUT` e `PKTS_IN`, mas agora se tratando da quantidade de pacotes. Os valores `sockstat-tcp-inuse` e `sockstat-udp-inuse` indicam a quantidade de pontos de comunicação TCP e UDP, respectivamente, em um determinado instante de tempo. O valor `tcp-conn-established` indica a quantidade de conexões TCP estabelecidas. Como os dados coletados pela ferramenta Performance Co-Pilot não estão disponíveis através do Ganglia, a integração de dados mostrada na Figura 5.2 permite ao administrador uma visão mais completa do estado do *cluster*.

Utilização de CPU

A utilização de CPU é uma importante variável quando se monitora as máquinas de um *cluster*. Através dela, é possível detectar possíveis sobrecargas na utilização de determinadas máquinas e a partir disso tomar uma decisão para melhorar a utilização do *cluster* como um todo.

A Figura 5.3 mostra a integração de dados coletados pelo Ganglia e pelo PCP durante um determinado período de monitoramento. Os valores `CPU_SYSTEM`, `CPU_IDLE` e `CPU_USER` foram coletados pelo Ganglia enquanto que o valor `all-load-1minute` foi coletado pelo PCP. O primeiro valor coletado pelo Ganglia, `CPU_SYSTEM`, indica a quantidade de CPU utilizada pelo sistema operacional. Os valores `CPU_IDLE` e `CPU_USER` indicam a quantidade de CPU não utilizada e a quantidade de CPU utilizada por aplicações de usuário, respectivamente. O valor coletado pelo PCP, `all-load-1minute` indica a média de de utilização de CPU durante um minuto. Embora a informação coletada pelo PCP possa ser obtida através do Ganglia, a Figura 5.3 mostra a informação de média de utilização de CPU coletada por intervalos menores em relação aos intervalos utili-

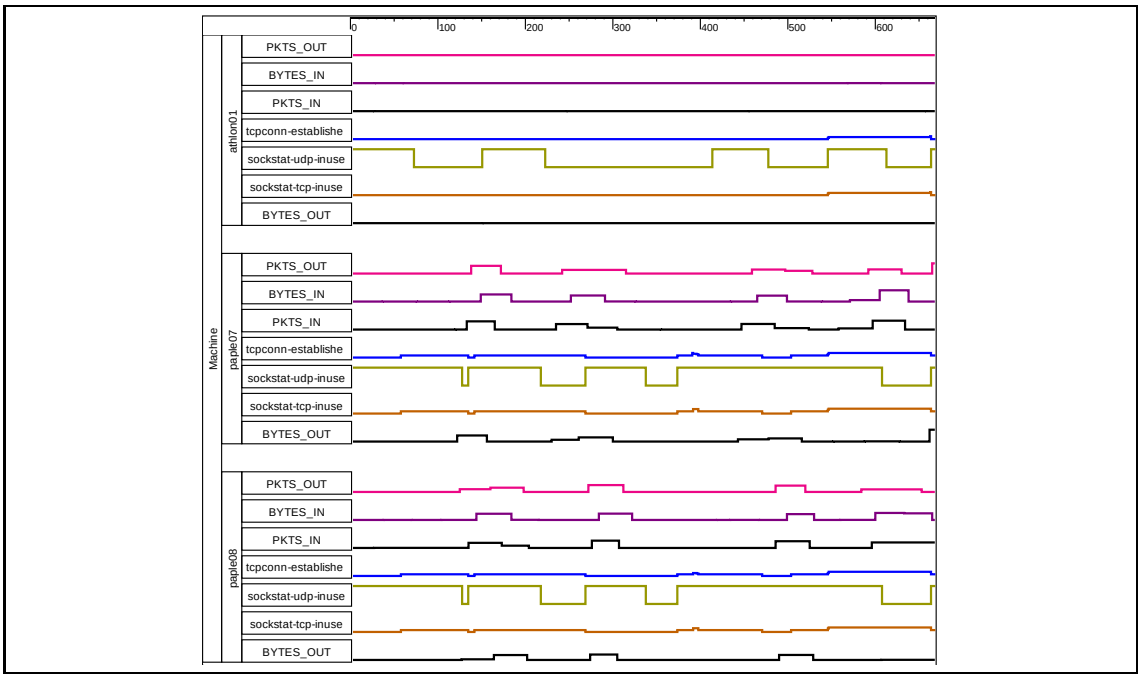

Figura 5.2: Visualização integrada em Pajé de informações de utilização de rede coletadas pelo Ganglia e pelo Performance Co-Pilot

zados para coletar as informações do Ganglia. Isso permite se obter informações mais instantaneamente, já que o Ganglia utiliza intervalos de coleta aleatórios, enquanto que o Performance Co-Pilot permite especificar o intervalo de coleta até o mínimo de um segundo.

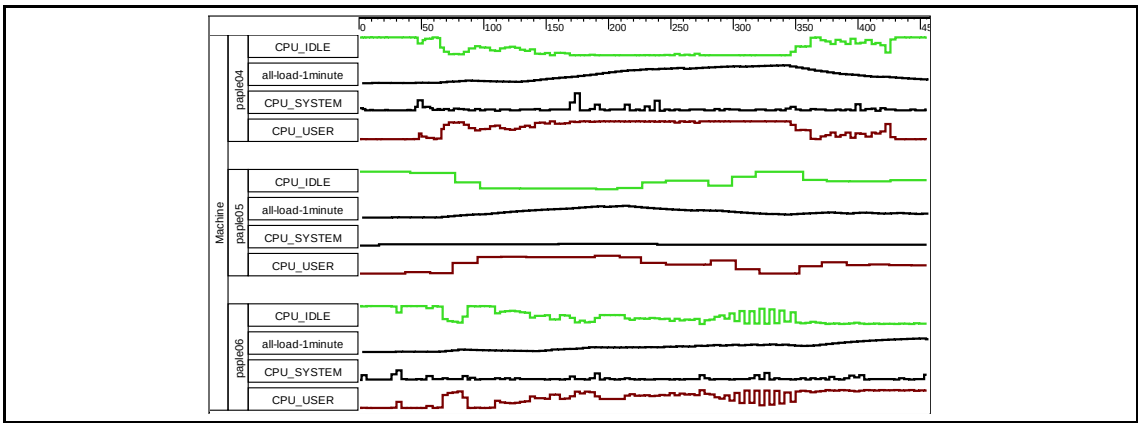

Figura 5.3: Visualização integrada em Pajé de informações de utilização de CPU coletadas pelo Ganglia e pelo Performance Co-Pilot

Utilização de memória

A Figura 5.4 mostra a integração de informações referentes a utilização de memória coletadas pelo Ganglia e pelo Performance Co-Pilot. A quantidade de memória livre, `MEM_FREE`, foi coletada pelo Ganglia. As outras informações, `mem-inactive`, `mem-active` e `mem-freemem` foram coletadas pelo Performance Co-Pilot. A primeira das informações

coletadas pelo PCP, `mem-inactive`, indica a quantidade de páginas de memória marcadas como inativas e que são candidatas para o descarte. O valor `mem-active` indica a quantidade de páginas de memória ativas e que foram recentemente referenciadas. A última informação do PCP, `mem-freemem` indica a memória livre de determinada máquina e é equivalente a informação `MEM_FREE` coletada pelo Ganglia. Os retângulos pontilhados na Figura 5.4 indicam o resultado da diferença do intervalo de coleta entre as ferramentas na obtenção da quantidade de memória livre em cada máquina. Como o PCP coletou os dados dessa figura em intervalos de um em um segundo, as alterações nos valores monitorados por ele são percebidas mais rapidamente que na ferramenta Ganglia.

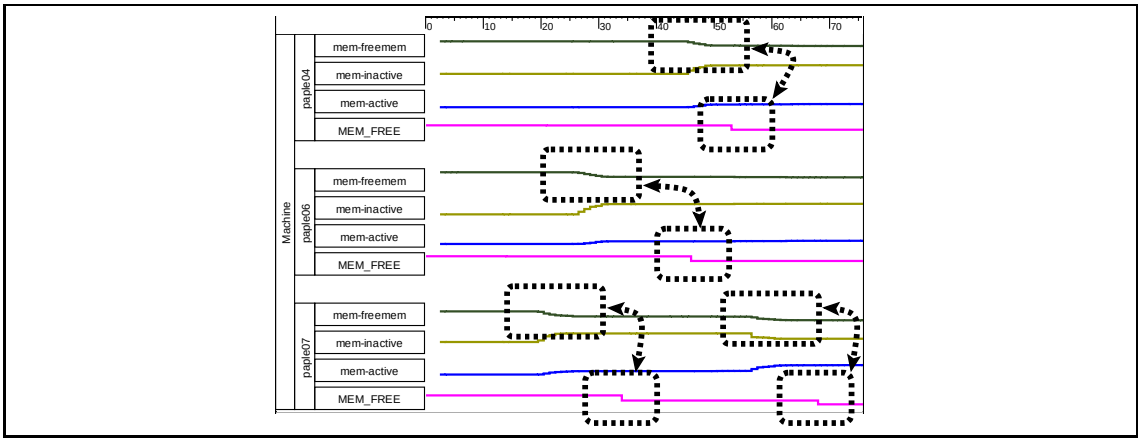

Figura 5.4: Visualização integrada em Pajé de dados de utilização de memória coletados pelo Ganglia e pelo Performance Co-Pilot

Monitoramento de processos

O monitoramento de processos é feito pelas ferramentas de monitoramento pela classificação dos processos de cada máquina em estados como bloqueado ou executando. Além dessa classificação, pode-se obter a quantidade de processos total no sistema e aqueles que estão executando.

A Figura 5.5 mostra a visualização integrada em Pajé de informações de monitoramento de processos coletados pelo Ganglia e pelo Performance Co-Pilot. Os dados `PROC_RUN` e `PROC_TOTAL` foram coletados pelo Ganglia. O Performance Co-Pilot foi utilizado para se coletar os dados `runq-sleeping`, `runq-runnable` e `runq-blocked`. O valor `PROC_RUN` indica os processos que estão em estado de execução em determinada máquina. O valor `PROC_TOTAL` indica a quantidade total de processos. Os valores coletados pelo PCP acima citados indicam, respectivamente, os processos aguardando execução, os processos prontos para executar e os processos bloqueados. A Figura 5.5, assim como a Figura 5.2, mostra dados complementares entre as informações coletadas pela ferramenta Ganglia e pelo Performance Co-Pilot.

Visualização integrada para monitoramento de cluster

Os tópicos anteriores abordaram visualizações integradas de informações coletadas pelo Ganglia e pelo PCP diferenciando-as de acordo com o tipo de informação que foi coletada. Nesta seção será apresentada a combinação de diferentes tipos de dados no processo de visualização integrada de informações.

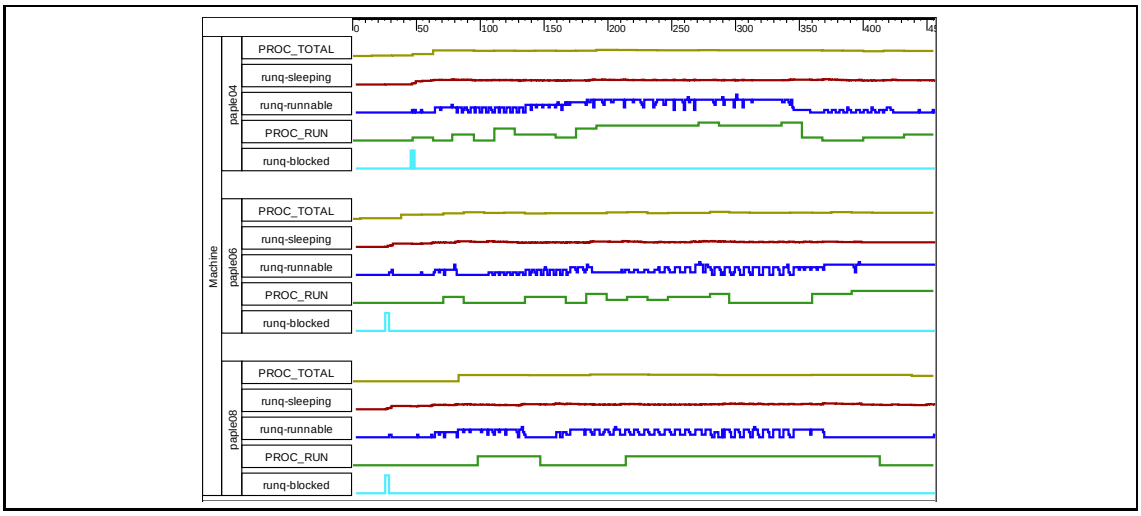

Figura 5.5: Visualização integrada em Pajé de dados de monitoramento de processos das ferramentas Ganglia e Performance Co-Pilot

A Figura 5.6 mostra os dados que foram coletados durante um período de monitoramento de duas máquinas que fazem parte do *cluster*, `paple02` e `paple06`. As informações `BYTES_OUT`, `BYTES_IN` e `CPU_USER` foram coletadas pela ferramenta Ganglia, enquanto que as outras foram coletadas pelo Performance Co-Pilot. Os retângulos pontilhados mostram uma possível relação entre as informações coletadas nas duas máquinas. Essas informações são `sockstat-tcp-inuse`, `mem-freemem`, `tcpconn-established` e `proc-nprocs`, que indica o número de processos em uma determinada máquina. Comparando os valores de `CPU_USER` entre as duas máquinas, percebe-se que a máquina `paple06` tem uma maior utilização. Isso se confirma através da informação coletada pelo PCP `all-load-15minute` que é maior na `paple06` em relação a máquina `paple02`. Essa informação indica a média de utilização de CPU nos últimos quinze minutos.

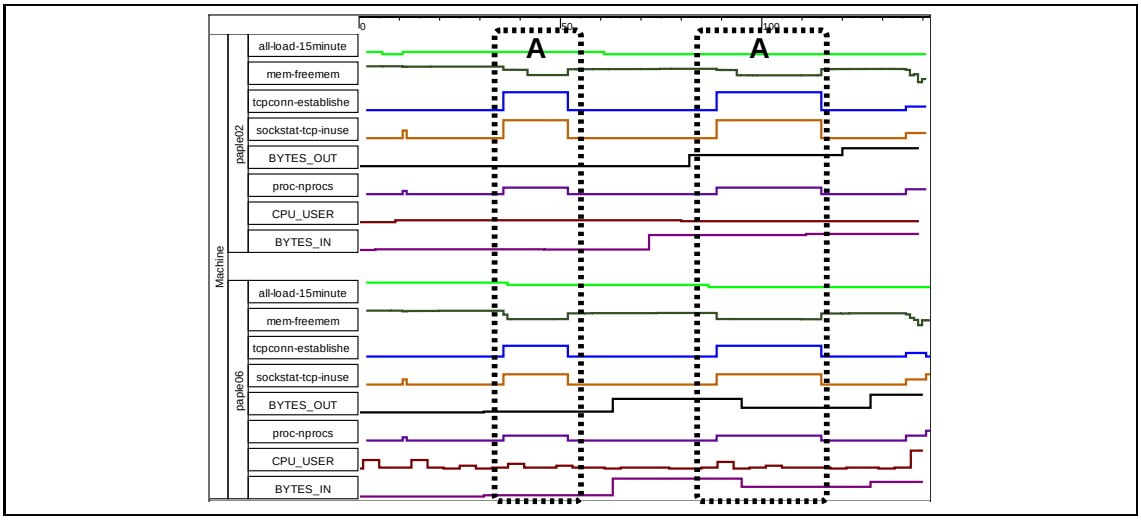

Figura 5.6: Visualização integrada em Pajé de dados de monitoramento de *cluster*. Nesta figura, informações de utilização de CPU, memória, rede e número de processos coletadas pelo Ganglia e Performance Co-Pilot estão visualizadas conjuntamente

Visualização integrada para monitoramento de Grid

No capítulo 2, foi mostrada uma situação onde existem múltiplos domínios administrativos cada qual com seu monitorador específico. Para mostrar a adaptação do protótipo a este tipo de situação, foram separadas quatro máquinas de uma rede, sendo que duas delas são monitoradas pela ferramenta Ganglia e as outras duas pelo Performance Co-Pilot. A Figura 5.7 mostra a visualização integrada dos dados coletados por essas duas ferramentas nos dois domínios administrativos. As máquinas `paple07` e `paple08` são monitoradas pela ferramenta Ganglia, representando um domínio administrativo. E as máquinas `paple02` e `paple03` são monitoradas pelo Performance Co-Pilot, representando o outro domínio administrativo.

Esse tipo de visualização integrada pode facilitar também na depuração de uma aplicação de larga escala quando esta é executando em ambientes multi-*cluster*ou em *Grids*. Para fazer isso, basta coletar os dados registrados nas máquinas de cada domínio administrativo desses ambientes com as informações de rastreamento da aplicação.

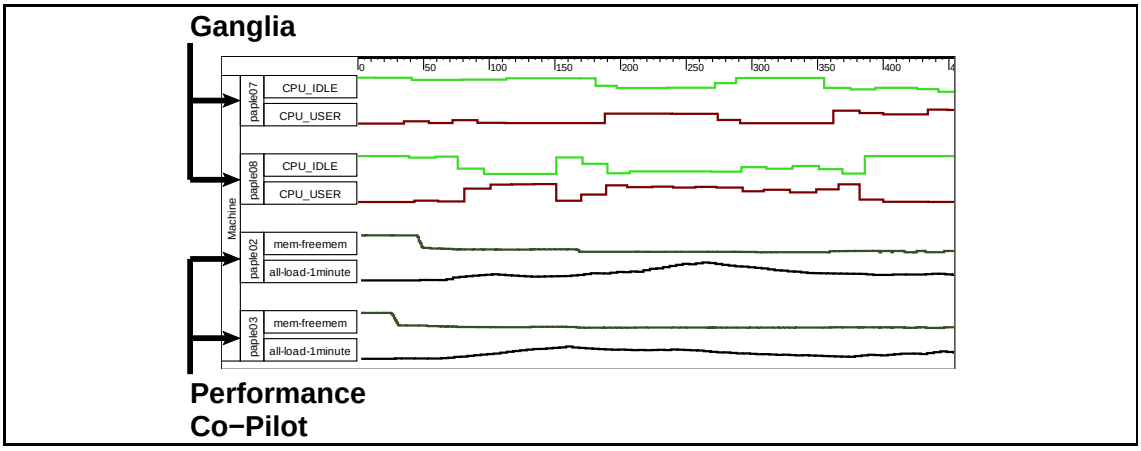

Figura 5.7: Visualização integrada em Pajé de informações coletadas em um ambiente semelhante a diferentes domínios administrativos de um *Grid*, cada qual monitorado por uma ferramenta de coleta de dados diferente

5.2.2 Depuração de aplicações DECK

No capítulo 4, seção 4.1, foram vistos o coletor de dados da biblioteca DECK, do sistema operacional Linux e das ferramentas de monitoramento Ganglia e Performance Co-Pilot. Foi visto que a coleta de dados de uma aplicação DECK é feita através do rastreamento do conjunto de funções da biblioteca que realiza comunicação coletiva, gerenciamento de fluxos de execução e comunicação através de caixas de mensagem.

Nesta seção serão mostrados a visualização de dados integrados dessas diferentes fontes de informação. Inicialmente, serão mostradas as funções de comunicação coletiva, depois exemplos de integração com utilização das funções de gerenciamento de fluxos de execução e exemplos com a utilização de funções para comunicação através de caixas de mensagens. No final, é mostrada a integração de dados na visualização do comportamento de uma aplicação *ping-pong*.

Comunicação coletiva

A comunicação coletiva no DECK é utilizada por aplicações onde seus processos podem fazer parte de um grupo. Ela facilita a programação oferecendo, por exemplo, serviços de *broadcast*, *scatter* e *gather* entre um grupo de processos. Nesta seção serão mostrados dois exemplos de integração de dados utilizando aplicações que utilizaram a comunicação coletiva do DECK.

A Figura 5.8 mostra a visualização integrada de dados de uma aplicação DECK com cinco processos. Essa aplicação utilizou as funções `deck_group_create`, para a criação de um grupo, e `deck_collective_bcast`, para o envio de uma mensagem. A mensagem tem origem no processo que executou na `paple02` para os processos que executaram nas outras máquinas, `paple03`, `paple05`, `paple06` e `paple07`. Os estados para cada uma dessas funções estão indicados na figura assim como as informações que foram coletadas pelas ferramentas Ganglia, `CPU_USER`, Performance Co-Pilot, `sockstat-tcp-inuse`, e Sistema operacional Linux, `OSState`. Nesta figura, observa-se o aumento da quantidade de conexões TCP em uso, `sockstat-tcp-inuse`, em todas as máquinas. Esse aumento aconteceu em instantes diferentes durante a execução da aplicação DECK. Isso acontece porque a coleta desse dado em cada máquina é feita por coletores do Performance Co-Pilot independentes com intervalos de coleta independentes. Outro fator que pode ser observado é o valor maior de utilização de CPU, `CPU_USER`, na máquina `paple06`. Caso essa aplicação DECK necessite de bastante processamento, esse valor pode influenciar negativamente o desempenho da execução da aplicação. O último fator que pode ser observado nesta figura é a quantidade de trocas de contexto do processo da aplicação que executou na máquina `paple02`. De acordo com a visualização, pode-se identificar na barra `OSState` da máquina `paple02` quatro blocos onde houveram intensas trocas de contexto. Nesta figura, as flechas pontilhadas identificadas pela letra A mostram a relação entre o fim do envio dos dados e essa intensa troca de contexto no nível do sistema operacional do processo que enviou as informações. Com isso, pode-se verificar que embora seja uma comunicação *broadcast*, os dados são enviados a cada membro do grupo um depois do outro.

A Figura 5.9 mostra o comportamento de uma aplicação DECK com cinco processos. Esses processos realizam operações de *reduce* ao longo de sua execução. Nesta figura, estão salientadas através dos retângulos pontilhados quatro dessas operações. A integração dos dados da aplicação foi feita com informações coletadas pelo Performance Co-Pilot, `proc-nprocs` e `sockstat-tcp-inuse`. A primeira informação diz a quantidade de processos que estão em determinada máquina. A segunda diz a quantidade de conexões TCP em uso em um determinado instante de tempo. Na figura, observa-se que em cada máquina existem diferentes valores tanto para a quantidade de processos quanto para a quantidade de conexões TCP em uso. Levando-se em conta que as máquinas são semelhantes tanto em *hardware* quanto em *software*, verifica-se que podem existir outros processos competindo com a execução dessa aplicação DECK. Essa figura também mostra a flexibilidade na integração dos dados, já que mostra apenas informações do Performance Co-Pilot com dados do DECK, sem dados do sistema operacional e do Ganglia. Isso é importante para oferecer ao programador a possibilidade de diminuir a quantidade de ferramentas de coleta utilizadas na etapa de coleta de dados. Através disso, o programador tem a possibilidade de diminuir a intrusão causada pelas diferentes ferramentas de coleta utilizadas e realizar uma análise mais próxima do comportamento real da aplicação, sem ferramentas de coleta.

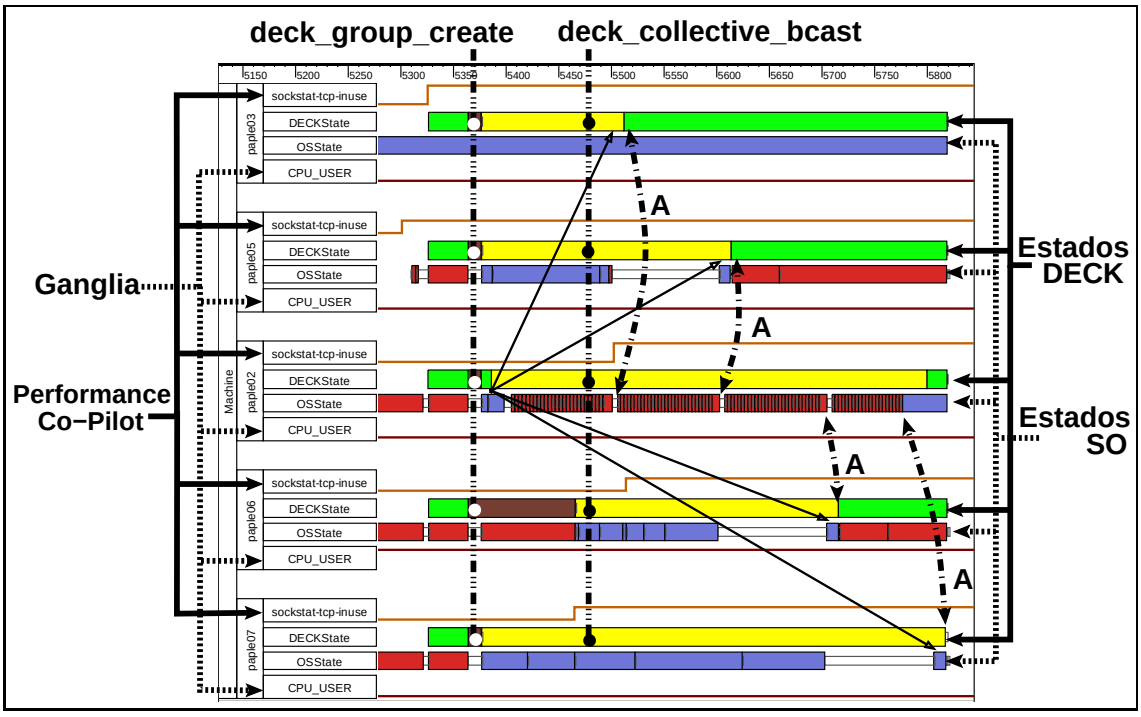

Figura 5.8: Visualização de dados das fontes de informação Ganglia, Performance Co-Pilot, aplicação DECK e sistema operacional. A figura mostra o comportamento de uma aplicação DECK com cinco processos em cinco máquinas diferentes

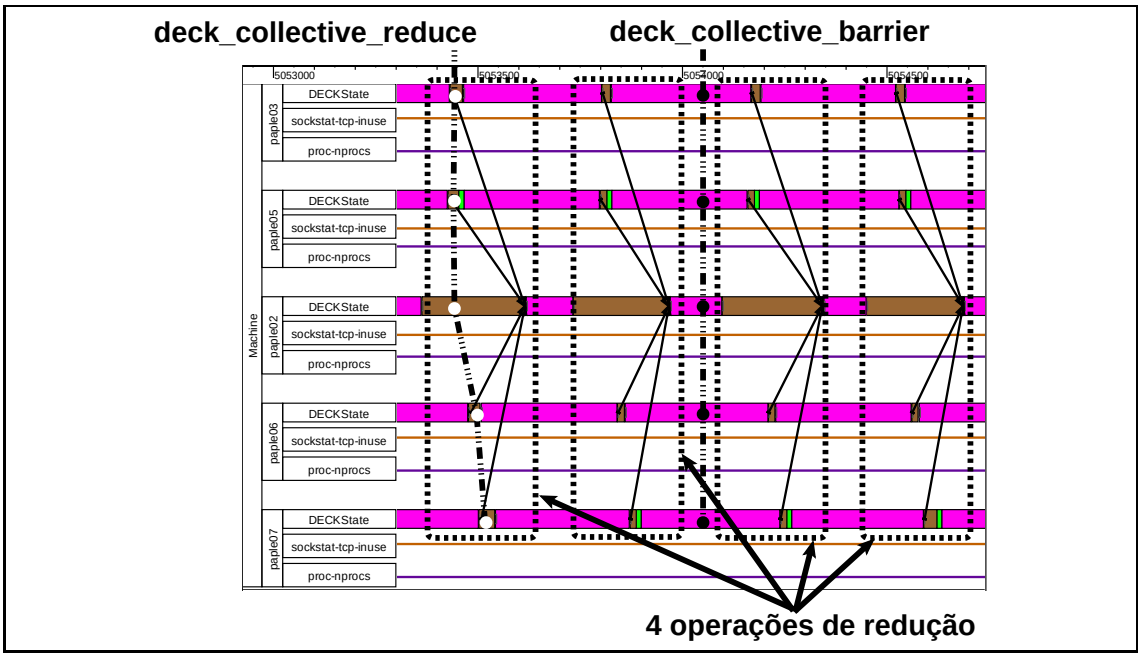

Figura 5.9: Visualização de dados das fontes de informação Performance Co-Pilot e biblioteca DECK. A figura mostra quatro execuções das operações de `deck_collective_reduce` e `deck_collective_barrier` em uma aplicação com cinco processos em cinco máquinas diferentes

Fluxos de execução

A biblioteca DECK permite que as aplicações explorem o paralelismo intra-nó através da utilização de funções de gerenciamento de fluxos de execução. A integração de dados realizada permite que os dados dos múltiplos fluxos de execução de um determinado nó sejam agrupados com as informações coletadas na mesma máquina por outras fontes de dados. Esta seção apresenta a visualização integrada dessas informações.

A Figura 5.10 mostra o comportamento da execução de uma aplicação DECK com três fluxos de execução em uma mesma máquina. As informações que foram coletadas através do rastreamento dessa aplicação DECK tanto no nível da biblioteca DECK quanto no nível do sistema operacional estão visualizadas concomitantemente com dados coletados pelo Performance Co-Pilot: `proc-nprocs`, `all-load-1minute` e `sockstat-tcp-inuse`. O primeiro indica o número de processos da máquina, o segundo a média da carga de processamento em 1 minuto e o terceiro a quantidade de conexões TCP em uso. As flechas no retângulo pontilhado A mostram a execução da função `deck_thread_create`. Essas flechas apontam do fluxo de execução criador para os fluxos de execução criados. A região B indica o momento que a alteração no número de processos da máquina foi observada pela ferramenta de monitoramento Performance Co-Pilot. Este atraso se deve ao intervalo de coleta dessa ferramenta, que é no mínimo de 1 em 1 segundos. Outro fato que pode-se observar nesta figura é o comportamento dos fluxos de execução criados no nível do sistema operacional. Embora a máquina `paple06` seja composta de dois processadores, o rastreamento desses dois fluxos de execução mostra que somente um deles estava executando por vez. Além disso, pode-se perceber que o estado do fluxo de execução principal no nível do sistema operacional está bloqueado o tempo inteiro.

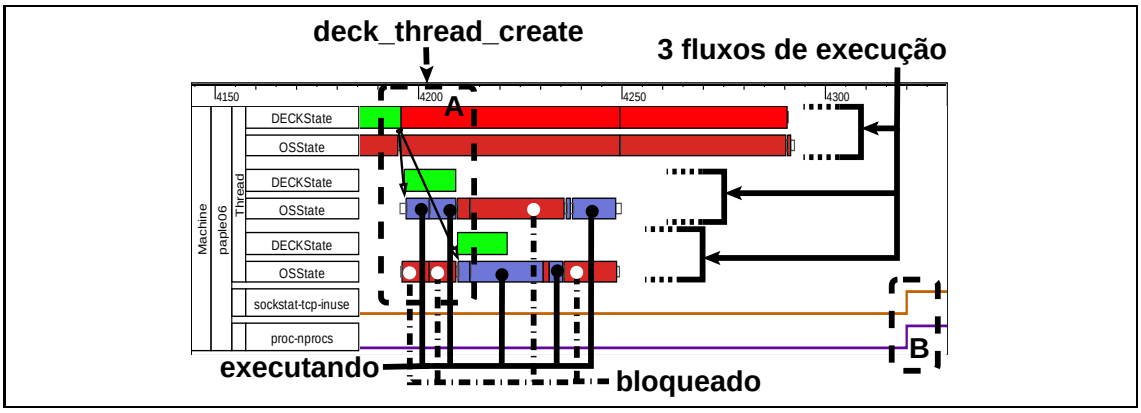

Figura 5.10: Visualização de dados de rastreamento de uma aplicação DECK com três fluxos de execução juntamente com informações do Performance Co-Pilot

Comunicação através de caixas de mensagem

A biblioteca DECK oferece ao programador um conjunto de funções para realizar a comunicação ponto-a-ponto entre processos de uma aplicação. Esse conjunto de funções realiza a comunicação através da semântica de caixas de mensagem, onde um processo que deseja receber dados deve criar uma caixa, e os processos que querem enviar dados para alguém deve realizar a clonagem da caixa de mensagem do processo destino. Após a criação e clonagem das caixas de mensagem, o programador pode utilizar funções de envio e recepção síncronos tradicionais. Esta seção apresenta um exemplo de visualização

integrada de dados de uma aplicação DECK com dados externos a essa aplicação. O programa DECK utiliza funções de comunicação ponto-a-ponto.

A Figura 5.11 mostra a visualização de dados de uma aplicação DECK composta por dois fluxos de execução, cada um em máquinas diferentes. Os dados de rastreamento da aplicação estão visualizados nesta figura concomitantemente com informações do Ganglia, `BYTES_IN`, `BYTES_OUT` e `CPU_USER`, e do Performance Co-Pilot, `all-load-1minute`. Os dois primeiros dados do Ganglia indicam a quantidade de *bytes* que estão chegando e saindo na placa de rede de uma máquina em um determinado instante de tempo, respectivamente. O terceiro dado do Ganglia indica a utilização de CPU em um determinado instante de tempo e a o dado do PCP, `all-load-1minute`, indica a média de utilização de CPU em um minuto. O retângulo pontilhado A nesta figura mostra a criação e clonagem de caixas de mensagem através das funções `deck_mbox_create` e `deck_mbox_clone`, respectivamente. Cada processo da aplicação visualizada nesta figura cria uma caixa de mensagem, que é clonado pelo outro processo. A execução da função de recepção de dados, `deck_mbox_retrv` e o período no qual o processo ficou bloqueado neste estado estão indicados na figura. A execução do envio da mensagem com a função `deck_mbox_post` está apontado na figura e é representada graficamente pelas flechas de um processo ao outro. O primeiro envio acontece do processo que está executando na máquina `paple02` para o processo da máquina `paple03`. O segundo acontece do processo da `paple03` para o processo da `paple02`. Os retângulos pontilhados B mostram a relação entre o dado coletado pelo Ganglia `BYTES_OUT` com o envio de informações realizado pelo processo da mesma máquina. Na `paple02` percebe-se que a quantidade de *bytes* saindo desse máquina começou antes da comunicação do processo DECK. Isso pode ter acontecido pela utilização da rede por outras aplicações dessa máquina e que podem influenciar negativamente o comportamento da aplicação DECK em questão. O retângulo pontilhado C mostra a relação entre o dado `BYTES_IN` e a recepção de uma mensagem pelo processo DECK da mesma máquina. Outro fator que pode ser observado nessa figura é a relação entre os dados do Ganglia, `CPU_USER`, e do PCP, `all-load-1minute`, com a região da aplicação onde houve um processamento significativo, apontado na figura pelo estado `running`.

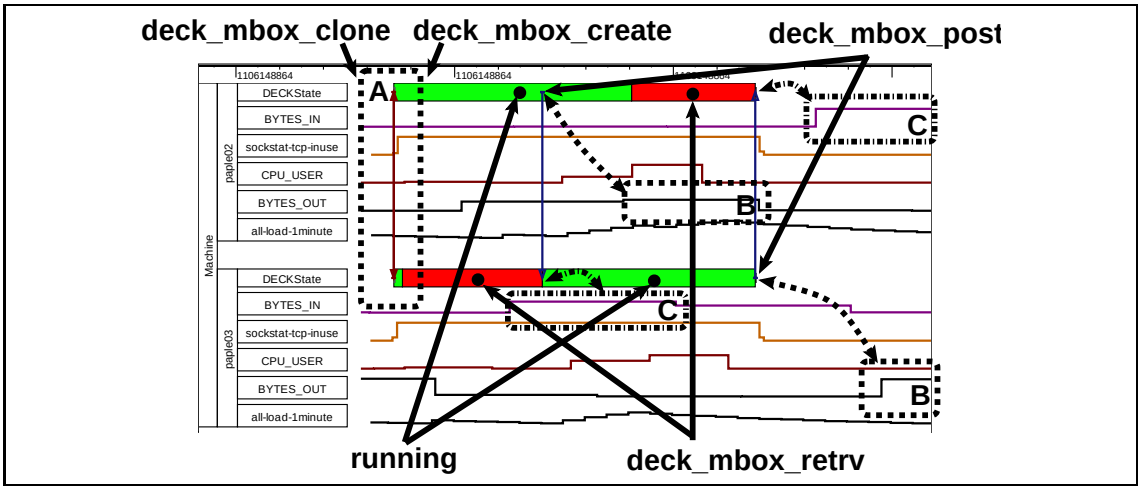

Figura 5.11: Visualização de dados coletados durante a execução de uma aplicação DECK com informações das ferramentas Ganglia e Performance Co-Pilot. A figura mostra dois fluxos de execução, cada um em máquinas diferentes, que utilizam funções `deck_mbox` para enviar e receber dados

Aplicação

Foi desenvolvida uma aplicação *ping-pong* para mostrar a análise do comportamento dessa aplicação através da visualização integrada de informações. A aplicação realiza quatro *ping-pongs* entre dois processos de uma aplicação DECK, sendo que entre cada recepção e envio de dados de um determinado processo é feito um processamento desses dados. Esse processamento utiliza bastante CPU para ser realizado.

A Figura 5.12.a mostra o comportamento da aplicação DECK *ping-pong*. As operações de *ping* realizadas pelo processo executado na `paple02` assim como as operações de *pong* realizadas pelo processo executado na `paple03` estão indicadas na figura. Além das informações da biblioteca DECK, a Figura 5.12.a mostra dados do Ganglia, `CPU_USER`, `BYTES_OUT` e `BYTES_IN`, e dados do Performance Co-Pilot, `all-load-1minute`. Os retângulos pontilhados identificados pela figura mostram a alteração nos valores de utilização de CPU, tanto absoluto no caso do `CPU_USER`, quanto uma média no caso do `all-load-1minute`. Essa alteração se deve ao processamento feito nos dados enviados pela operação de *ping*. O local na barra de estado do processo DECK da figura onde este processamento acontece está indicado pelos pontos pretos. Pode-se observar também na Figura 5.12.a a relação entre os valores de `BYTES_OUT` e `BYTES_IN` coletados pelo Ganglia com o momento onde são feitas as operações de *ping* e *pong* pelos processos. Essa relação no caso do valor `BYTES_OUT` está representado nesta figura pela letra B.

A Figura 5.12.b mostra o comportamento durante a execução da mesma aplicação *ping-pong*. No entanto, durante a coleta de dados para a visualização integrada das informações, foi adicionada uma utilização artificial da CPU através de aplicações simples com laços infinitos. Essa adição foi feita no início da execução da aplicação apenas na máquina `paple03` e depois, no fim da execução, apenas na máquina `paple02`. Os retângulos pontilhados A identificados na Figura 5.12.b mostram o resultado dessa utilização de CPU artificial. Através da ferramenta Pajé, pode-se verificar o tempo de duração de um estado na janela de visualização. Utilizando este recurso, pode-se verificar que o estado identificado pela letra C nas duas visualizações da Figura 5.12 teve um tempo de execução dois segundos maior em relação a execução sem aplicações concorrentes. Caso não fossem utilizadas dados externos a aplicação, seria mais complicado verificar o porquê desse comportamento diferenciado visto na Figura 5.12.b. Além disso, a utilização de CPU por aplicações concorrentes alterou o monitoramento das dados `BYTES_OUT` e `BYTES_IN`, como pode ser observando comparando-se as visualizações da Figura 5.12.

5.2.3 Depuração de aplicações MPI

No capítulo 4, seção 4.1, foram vistos o sistema de rastreamento de aplicações MPI e a coleta de dados do sistema operacional, Ganglia e Performance Co-Pilot. Na seção 4.3 deste mesmo capítulo foi vista a visualização esquemática que se deseja obter para a depuração de aplicações paralelas, tanto em DECK quanto em MPI. Nesta seção serão mostrados os resultados obtidos através do protótipo de integração na unificação de informações oriundas das fontes de informação acima citadas. Esses resultados serão apresentados através da figuras da ferramenta de visualização Pajé. Inicialmente, será apresentada uma comparação entre as ferramentas já existentes para as fontes de informação utilizadas e a visualização conjunta proporcionada por este trabalho. Depois serão mostradas a visualização em Pajé de duas aplicações: *ping-pong* e filtro de mediana.

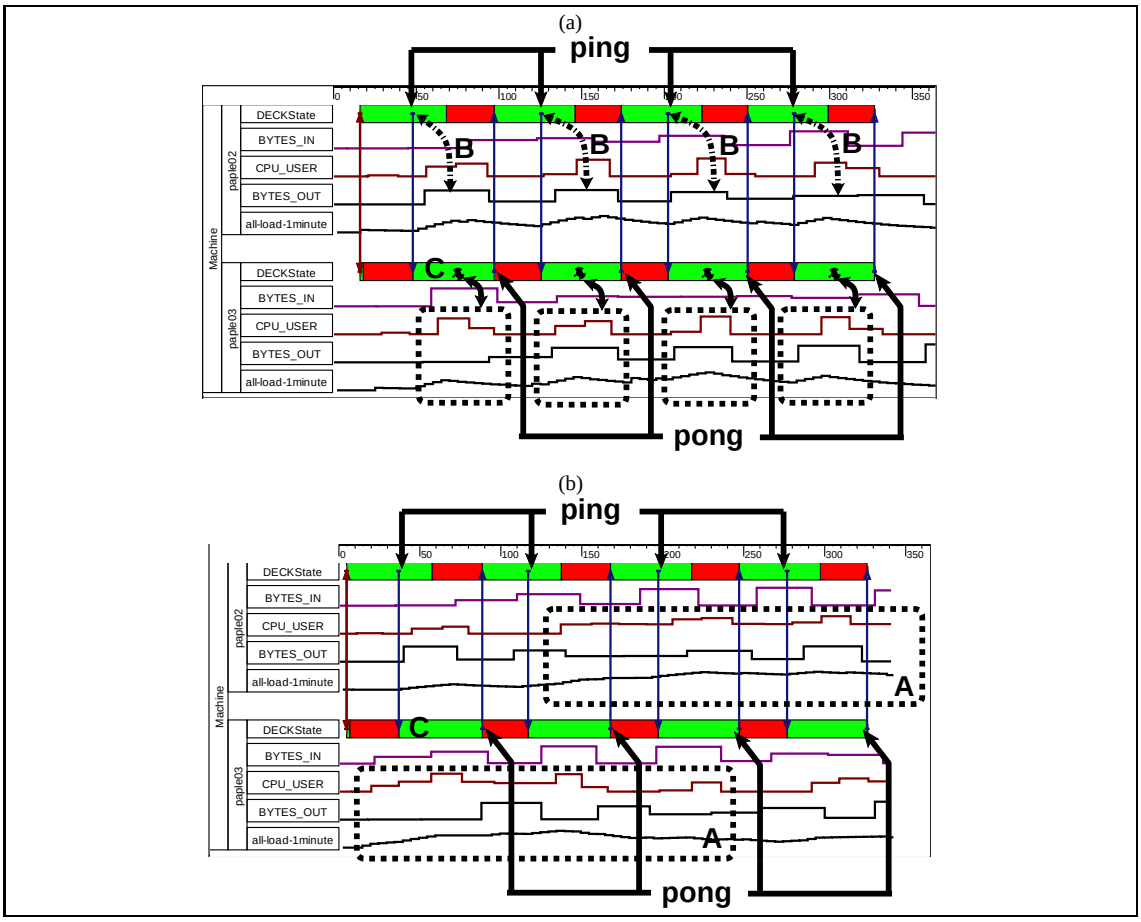

Figura 5.12: Visualização integrada em Pajé de informações para a depuração de uma aplicação DECK *ping-pong*. A figura inferior mostra o comportamento da aplicação quando executada com outras aplicações intrusivas. A figura superior mostra o comportamento da aplicação sem essa intrusão

Comparação

Foi desenvolvida uma aplicação que quando executada é composta por dois processos, sendo que processo principal realiza três comunicações com o outro através das funções `MPI_Send` e `MPI_Recv`. Essa aplicação foi monitorada tanto no nível do MPI, através da biblioteca MPE, quanto no nível do sistema operacional. O ambiente de execução dessa aplicação foi monitorado pelas ferramentas Ganglia e Performance Co-Pilot.

Normalmente, quando não há uma integração de dados, o programador que está analisando o comportamento de sua aplicação visualiza as informações registradas pelas ferramentas de coleta de forma independente. As Figuras 5.13 e 5.14 mostram a visualização independente de dados da aplicação MPI através da ferramenta Upshot e dados coletados pelo Ganglia visualizado através de uma interface para a internet, respectivamente. Como explicado no capítulo 2, essa visualização independente dificulta a correlação entre as informações e não permite uma eficiente análise quando necessita de informações do ambiente de execução para se analisar o comportamento de uma aplicação MPI.

A Figura 5.15 mostra a visualização integrada dos dados vistos independentemente nas figuras 5.13 e 5.14. Com essa visualização integrada, é possível perceber melhor as relações causais entre possíveis eventos externos a uma aplicação MPI com dados compor-

Figura 5.13: Visualização de uma aplicação MPI utilizando a ferramenta Upshot, que acompanha a distribuição da implementação MPICH

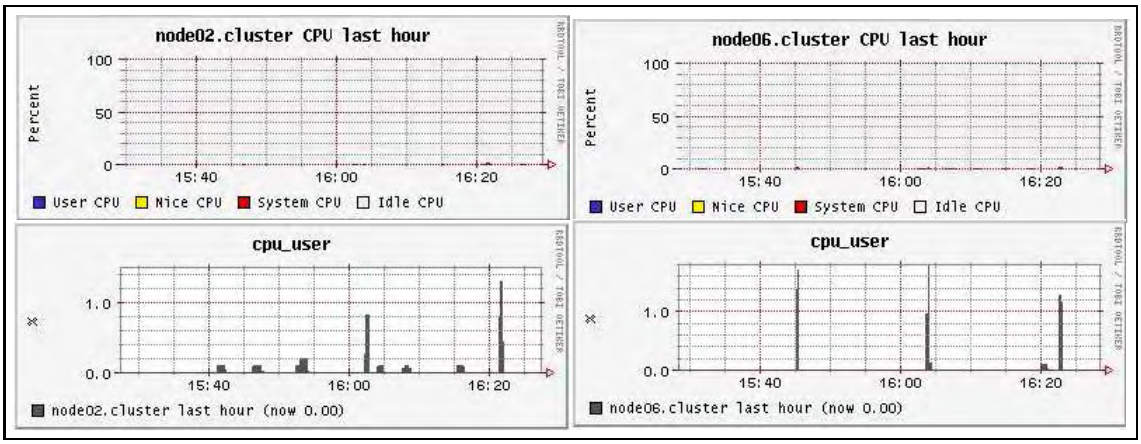

Figura 5.14: Visualização independente das informações coletadas pela ferramenta Ganglia. Os dados são visualizados utilizando uma interface web

tamentais coletados no nível da aplicação. Nesta figura, os dados coletados pelas quatro fontes de informação, Ganglia, PCP, sistema operacional e biblioteca de rastreamento do MPI estão indicados. Além disso, os estados `MPI_send`, `MPI_Recv` e `executando` do rastreamento do MPI estão indicados juntamente com os estados `bloqueado` e `executando` do nível do sistema operacional.

Aplicações

Foram desenvolvidas duas aplicações para mostrar o resultado do processo de integração feito pelo protótipo descrito no capítulo 4. A primeira aplicação é um *ping-pong*. A segunda é uma aplicação de filtro de mediana. A seguir é apresentada a visualização integrada de dados coletados durante a execução dessas aplicações.

A aplicação *ping-pong* construída em MPI é composta por dois processos. O processo principal envia ao outro uma mensagem e depois essa mensagem é retornada ao processo principal. Tanto o tamanho da mensagem quanto o número de vezes que a comunicação entre os dois processo vai acontecer é passado como parâmetro para a aplicação. A Figura 5.16 mostra a visualização de duas execuções dessa aplicação *ping-pong* com um tamanho de mensagem de 1 Megabyte. Na primeira execução enfatizada pelo retângulo pontilhado A desta figura tem-se a realização de três *ping-pongs*. Na segunda execução, no retângulo pontilhado B, tem-se a realização de cinco *ping-pongs*. A visualização das informações

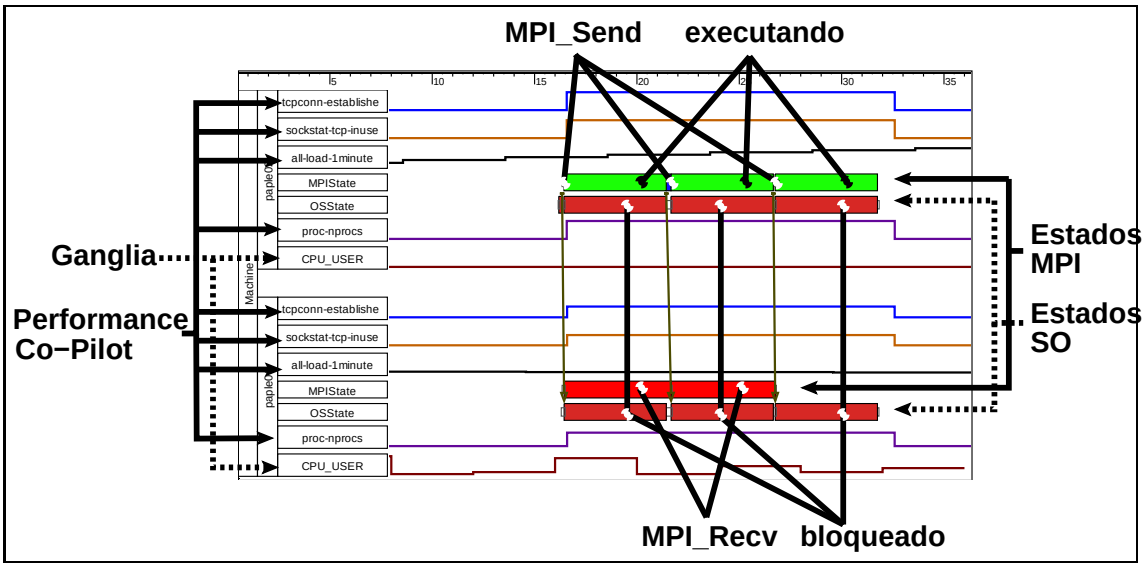

Figura 5.15: Visualização em Pajé da mesma aplicação da Figura 5.13, mas agora com a integração de dados oriundos do Sistema operacional Linux e das ferramentas de monitoramento Ganglia e Performance Co-Pilot

do estado do MPI e do sistema operacional Linux, visualizado nas barras inferiores de cada fluxo de execução da figura, está conjuntamente visualizado com informações do Ganglia, `CPU_USER`, e do Performance Co-Pilot, `tcpconn-establish` e `proc-nprocs`. As flechas indicadas pela letra C nesta figura apontam o momento de início da execução do processo MPI na máquina `paple02` e a relação com os dados coletados pela ferramenta Performance Co-Pilot, `tcpconn-establish` que indica a quantidade de conexões TCP estabelecidas e `proc-nprocs` que indica a quantidade de processos na máquina. Essa relação pode ser observada também na primeira execução da aplicação *ping-pong*.

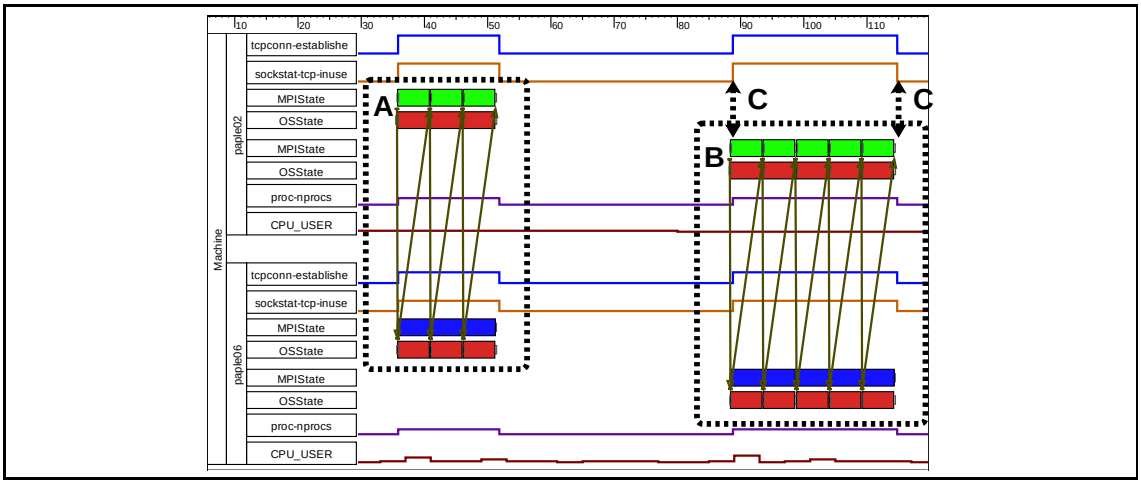

Figura 5.16: Visualização integrada com dados de monitoramento de duas execuções de uma aplicação MPI *ping-pong*

A outra aplicação construída foi um filtro de mediana. Esse tipo de filtro é utilizado em processamento de imagens para retirar ruídos do tipo sal e pimenta. Esse tipo de ruído se caracteriza por pontos pretos ou brancos distribuídos na imagem. Normalmente,

o tamanho desses pontos fica restrito ao tamanho de um *pixel* da imagem. O filtro de mediana utiliza uma máscara para calcular o novo valor para cada *pixel* de uma imagem. Esse cálculo consiste em verificar os valores dos *pixels* vizinhos e ordená-los. O novo valor do *pixel* que está sendo calculado será o valor mediano do ordenamento feito com os valores dos *pixels* vizinhos.

O filtro de mediana é uma aplicação considerada trivialmente paralelizável. A paralelização adotada na construção dessa aplicação consistiu em dividir a imagem entre os processos que farão o processamento. Na implementação, optou-se pelo processo mestre também realizar o processamento da imagem.

A Figura 5.17 mostra a visualização do comportamento durante a execução do filtro de mediana tanto no nível da aplicação quanto no ambiente de execução. Os dados do ambiente de execução nesta figura são `CPU_USER` e `all-load-1minute`, coletados respectivamente pelo Ganglia e pelo Performance Co-Pilot. A aplicação é composta por cinco processos, sendo que cada um é executado em uma máquina diferente. Na visualização vista nesta figura, o estado de execução do filtro mediana está indicado. O retângulo pontilhado A indica o momento onde as diferentes partes da imagem são enviadas do processo principal, executado na máquina `paple03`, para os outros processos. O início do estado de execução visualizado na barra `MPIState` está relacionado com o aumento de utilização de CPU, detectado tanto pela ferramenta Ganglia, através do dado `CPU_USER`, quanto pelo Performance Co-Pilot, através do dado `all-load-1minute`. O retângulo pontilhado B indica o momento onde as informações processadas por cada processo são enviadas ao processo mestre executando na máquina `paple03`. Percebe-se a diminuição da utilização de CPU no momento que o estado de execução da aplicação termina. O estado `MPI_Send`, observado no estado do processo executando na máquina `paple08`, é executado por todos os processo que enviaram os dados para o processo principal.

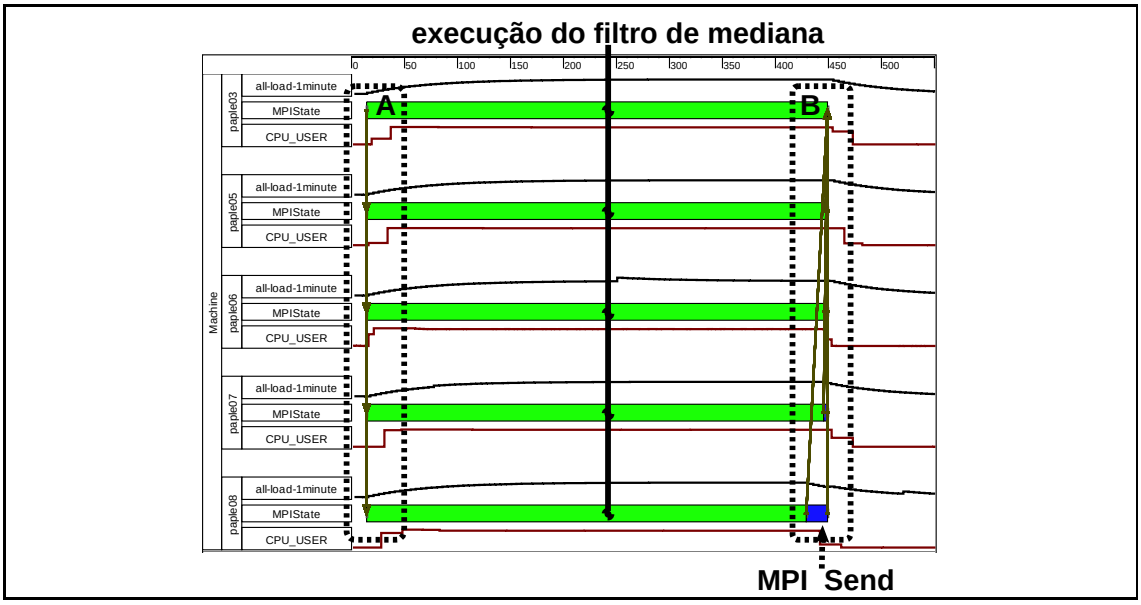

Figura 5.17: Visualização integrada com dados do ambiente de execução de uma aplicação que realiza o filtro de mediana sobre uma imagem

A Figura 5.18 mostra o início da execução da aplicação MPI filtro de mediana. Esta figura foi feita utilizando-se os recursos de *zoom* e utilização de filtros do Pajé e mostra o retângulo pontilhado A da Figura 5.17. Na Figura 5.18 tem-se apenas a visualização de

informações do nível da aplicação MPI e do nível do sistema operacional. Os estados que cada nível obteve durante a execução estão indicados na figura. O retângulo pontilhado A nesta figura mostra a relação entre a recepção das mensagens enviadas pelo processo na máquina `paple03` e as sucessivas trocas de contexto no nível do sistema operacional durante a recepção dessas mensagens. A flecha B indica a relação entre o início dessas sucessivas trocas de contexto no processo MPI da máquina `paple06` com o início da primeiro envio para esse processo. Percebe-se que antes desse primeiro envio de fato chegar no processo da máquina `paple06`, esse mesmo processo no nível do sistema operacional já estava trabalhando na recepção dessa mensagem, indicado na figura pelo retângulo pontilhado C.

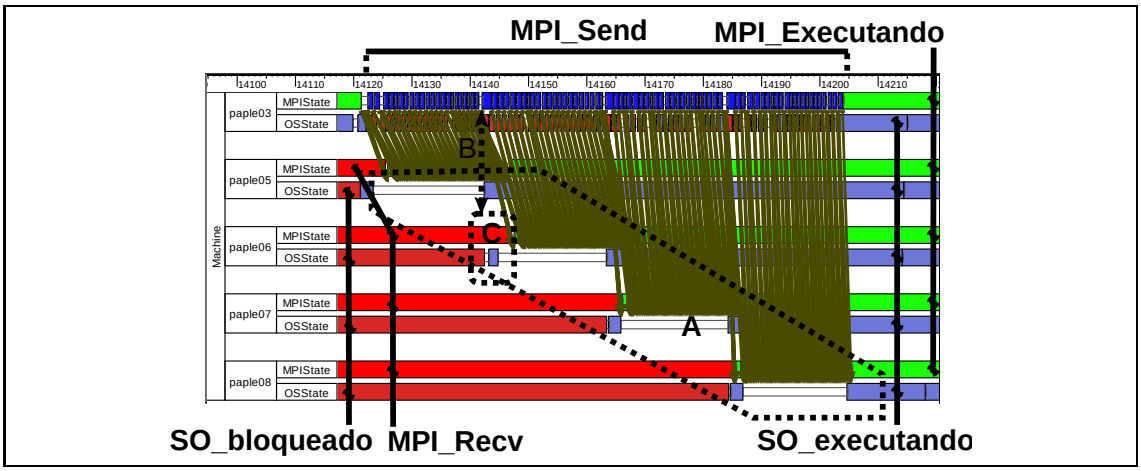

Figura 5.18: Visualização integrada com dados do sistema operacional do início da execução da aplicação filtro de mediana

A Figura 5.19 mostra o fim da execução da aplicação filtro de mediana com informações do sistema operacional e a quantidade de utilização de CPU coletada pela ferramenta Ganglia. Na figura, percebe-se que o processo que estava executando na máquina `paple08` ficou aguardando o fim do processamento de um dos pedaços da imagem no processo principal, executado na máquina `paple03`. Outro fator que pode ser observado nessa figura é o fim das sucessivas trocas de contexto durante a execução do processamento da imagem no nível da aplicação MPI. Isso está indicado na figura através das linhas verticais B. Como o processo na máquina `paple08` terminou antes que os outros, pode-se perceber a diminuição da utilização do dado `CPU_USER` coletado pelo Ganglia. A distância indicada pela flecha A mostra o atraso na coleta de dados do Ganglia, já que seu intervalo de coleta é aleatório para diminuir a intrusão durante o monitoramento do *cluster*.

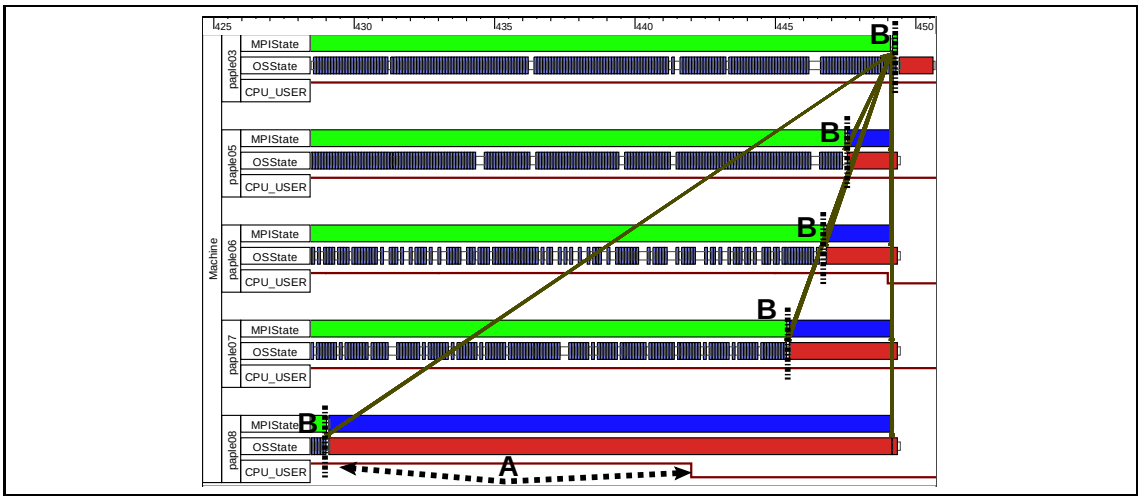

Figura 5.19: Visualização integrada com dados do sistema operacional e utilização de CPU coletada pelo Ganglia do fim da execução da aplicação de filtro de mediana

5.3 Conclusão do capítulo

Neste capítulo foram apresentados os resultados do processo de integração de dados através da ferramenta protótipo desenvolvida. Esses resultados foram divididos basicamente em três áreas: monitoramento de *cluster*, depuração de aplicações DECK e depuração de aplicações MPI. A primeira área tratou da integração de dados de diferentes ferramentas de monitoramento de *cluster*. Foi vista também nesta área uma visualização conjunta de informações onde um determinado conjunto de máquina era monitorado por uma ferramenta de *cluster* e outro conjunto por outra. A segunda área tratou da integração de dados originados do rastreamento de uma aplicação DECK com informações externas a aplicação. A última área tratou da integração de dados de rastreamento de uma aplicação MPI com dados do ambiente de execução.

No próximo capítulo serão apresentados as conclusões deste trabalho assim como algumas sugestões de trabalhos futuros.

6 CONCLUSÃO

A utilização de *cluster* de computadores para se obter alto desempenho na execução de aplicações tem aumentado ao longo do tempo. Isso pode ser verificado comparando sucessivos levantamentos das pesquisas realizadas que verificam quais são as 500 máquinas mais rápidas do mundo. Na listagem dessas máquinas de novembro de 2004, quase 60% das máquinas são classificadas como *cluster* (MEUER et al., 2004).

No entanto, a utilização de um *cluster* implica naturalmente no monitoramento de seus recursos de forma a auxiliar tanto o administrador quanto ao desenvolvedor de aplicações paralelas para essa arquitetura. Como visto no capítulo 2, algumas situações surgem durante o monitoramento dessa arquitetura que tornam esta tarefa complicada. Por exemplo, algumas vezes o administrador necessita de mais de uma ferramenta de monitoramento para realizar uma análise completa dos recursos e identificar possíveis problemas. Outras vezes, o desenvolvedor necessita de informações do ambiente de execução, coletadas por ferramentas de monitoramento, para melhor entender sua aplicação. Outra situação ainda acontece quando existe uma aplicação de larga escala a ser executada em um *Grid*, sendo que cada domínio administrativo desse *Grid* tem ferramentas de monitoramento diferentes e os dados coletados por elas devem ser integrados com os da aplicação para um melhor entendimento desta. Todas essas situações se caracterizam pela existência de múltiplas fontes de informação e pela independência entre essas fontes de dados.

Esse trabalho propôs um modelo de integração de dados para informações coletadas por diferentes ferramentas. Esse modelo, descrito no capítulo 3, é dividido em duas etapas. A primeira etapa consiste na coleta de informações por diferentes ferramentas de coleta. A segunda etapa consiste em integrar as informações coletadas pelas diferentes fontes de dados utilizadas na primeira etapa. A segunda parte é dividida em três subetapas que são: sincronização, unificação e padronização dos dados. O resultado desse processo é a integração de dados para a visualização conjunta. Para validar este modelo, foi construída uma aplicação protótipo que realiza a integração de informações das ferramentas de monitoramento Ganglia e Performance Co-Pilot, além de informações de rastreamento de aplicação DECK, MPI e dados das trocas de estado do Sistema Operacional Linux.

Os resultados obtidos com a utilização do modelo de integração de dados proposto neste trabalho podem ser usados tanto na área de monitoramento de *cluster* quanto na área de depuração de aplicações paralelas. Uma vantagem da utilização desse modelo é que ele é flexível a ponto de não especificar a utilização de uma determinada ferramenta de coleta de dados. Dessa forma, podem ser consideradas ferramentas de monitoramento e bibliotecas de rastreamento já utilizadas por administradores e desenvolvedores de aplicações paralelas. Outra vantagem é que o modelo pode se adaptar também a outras ferramentas de visualização bastando apenas que se faça uma padronização de dados relacionada a

ferramenta de visualização escolhida.

Dentre os resultados obtidos através da utilização do protótipo, foi visto que é possível com o protótipo integrar informações de diferentes ferramentas de monitoramento para a visualização simultânea. Além disso, o protótipo foi utilizado para se poder visualizar aplicações paralelas desenvolvidas com a biblioteca DECK e uma implementação em MPI. Nesses dois casos, os dados de rastreamento das aplicações puderam ser integrados com informações do ambiente de execução, coletados ou pelas ferramentas de monitoramento, ou no nível do sistema operacional. Foi visto também uma comparação com visualizações independentes de dados de uma aplicação MPI com o Upshot e dados do Ganglia com uma interface Web comparada com a visualização integrada em Pajé proporcionada pelo protótipo. Nessa comparação, pode-se obter uma melhor correlação dos dados do MPI com dados coletados pelo Ganglia através da utilização do protótipo de integração de dados. Além desses resultados, pode-se mostrar uma situação semelhante a um *Grid* com dois domínios administrativos, sendo que cada um desses domínios é monitorado por uma ferramenta diferente. A visualização das informações oriundas de diferentes domínios administrativos foi feita através da utilização do protótipo para a integração de dados.

Como sugestões de trabalhos futuros, tem-se a adição de novas fontes de informação ao protótipo, de forma que ele possa ser utilizado com outras configurações de *clusters* e ambientes de desenvolvimento. Dentre essas novas fontes, vale citar os dados coletados através da interface JVMTI, do Java, e também dados que podem ser coletados de algum gerenciador de processos para *cluster*, como por exemplo o PBS (HENDERSON, 1995). Outra sugestão de trabalhos futuros é a alteração do modelo e do protótipo de forma que o sistema de integração possa ser utilizada de forma *online*. Essa alteração beneficiaria principalmente a área de monitoramento de *cluster*, fazendo com que os dados de monitoramento coletados por ferramentas diferentes fossem sendo visualizados no momento em que são coletados, e não depois do fim do período de coleta de dados. Ferramentas como o Ganglia e o Performance Co-Pilot, utilizadas neste trabalho, já utilizam um fluxo constante de dados coletados. Dentre esses trabalhos futuros, está em desenvolvimento o trabalho de rastreamento de aplicações Java através da interface JVMTI (WIEDENHOFT; STEIN, 2005). No caso da utilização do modelo de forma *online*, existe um trabalho que procura transformar o Pajé, que é uma ferramenta de visualização genérica *offline*, em uma ferramenta que se adapte a situações onde existe um fluxo contínuo de dados.

REFERÊNCIAS

AGARWALA, S.; POELLABAUER, C.; KONG, J.; SCHWAN, K.; WOLF, M. System-Level Resource Monitoring in High-Performance Computing Environments. **Grid Computing**, Seatle, USA, v.3, n.1, p.273–289, 2003.

BAKER, M.; SMITH, G. **GridRM**: a resource monitoring architecture for the grid. [S.l.: s.n.], 2002. p. 268–273. (Lecture Notes in Computer Science, v.2536).

BARRETO, M.; RIVIERE, M.; NAVAUX, P. DECK: a new model for a distributed executive kernel integrating communication and multithreading for support of distributed object oriented application with fault tolerance support. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACION, 4., 1998, Neuquém, Argentina. **Trabajos Seleccionados...** [S.l.: s.n.], 1998. v.2, p.623–637.

BROBERG, M.; LUNDBERG, L.; GRAHN, H. Visualization and Performance Prediction of Multithreaded Solaris Programs by Tracing Kernel Threads. In: INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM, 13.; SYMPOSIUM ON PARALLEL AND DISTRIBUTED PROCESSING, 10., 1999, San Juan, Puerto Rico. **Proceedings...** Berlin: Springer-Verlag, 1999. p.407.

BRUNST, H.; HOPPE, H.-C.; NAGEL, W. E.; WINKLER, M. Performance Optimization for Large Scale Computing: the scalable vampir approach. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE-PART II, 2001. **Proceedings...** [S.l.]: Springer-Verlag, 2001. p.751–760.

BUYYA, R. Parmon: a portable and scalable monitoring system for clusters. **Software-Practice and Experience**, [S.l.], v.30, n.7, p.723–739, 2000.

BUYYA, R. (Ed.). **High Performance Cluster Computing**. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

CASSALI, R.; BARRETO, M. E.; ÁVILA, R. B.; NAVAUX, P. O. A. Group Communication Service for DECK. In: CONFERENCIA LATINOAMERICANA DE INFORMATICA, 26., 2000, México. **Proceedings...** México: TEC de Monterrey, 2000. CD-ROM.

DUMANT, B.; HORN, F.; TRAN, F. D.; STÉFANI, J.-B. Jonathan: an open distributed processing environment in java. In: IFIP INTERNATIONAL CONFERENCE ON DISTRIBUTED SYSTEMS PLATFORMS AND OPEN DISTRIBUTED PROCESSING, MIDDLEWARE, 1998, The Lake District, England. **Proceedings...** [S.l.: s.n.], 1998. p.173–190.

FERRETO, T. C.; ROSE, C. A. F. de; ROSE, L. de. RVision: an open and high configurable tool for cluster monitoring. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, CCGRID, 2., 2002. **Proceedings...** [S.l.]: IEEE Computer Society, 2002. p.75.

FERRETO, T.; ROSE, C. de. Improving Performance Analysis Using Resource Management Information. In: INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING, 2003, Hyderabad, India. **Proceedings...** Heidelberg: Springer-Verlag, 2003. p.449–458.

GROPP, W.; LUSK, E.; DOSS, N.; SKJELLUM, A. High-performance, portable implementation of the MPI Message Passing Interface Standard. **Parallel Computing**, Amsterdam, The Netherlands, v.22, n.6, p.789–828, Sept. 1996.

HENDERSON, R. L. Job Scheduling Under the Portable Batch System. In: WORKSHOP ON JOB SCHEDULING STRATEGIES FOR PARALLEL PROCESSING, IPPS, 1995. **Proceedings...** Heidelberg: Springer-Verlag, 1995. p.279–294.

KARAVANIC, K. L.; MYLLYMAKI, J.; LIVNY, M.; MILLER, B. P. Integrated visualization of parallel program performance data. **Parallel Computing**, Amsterdam, The Netherlands, v.23, n.1–2, p.181–198, 1997.

KERGOMMEAUX, J. C. de; OLIVEIRA STEIN, B. de. Flexible performance visualization of parallel and distributed applications. **Future Generation Computer Systems**, Amsterdam, The Netherlands, v.19, n.5, p.735–747, 2003.

KERGOMMEAUX, J. C. de; VINCENT, J.-M.; OTTOGALLI, F.-G. Exploring Performances of Parallel Applications on Clusters. In: WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS, SCI, 7., 2003, Orlando, Florida, USA. **Proceedings...** [S.l.: s.n.], 2003.

KOPETZ, H.; OCHSENREITER, W. Clock synchronization in distributed real-time systems. **IEEE Transactions on Computers**, Washington, DC, USA, v.36, n.8, p.933–940, 1987.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. **Communications of the ACM**, New York, NY, USA, v.21, n.7, p.558–565, 1978.

LIANG, Z.; SUN, Y.; WANG, C.-L. ClusterProbe: an open, flexible and scalable cluster monitoring tool. In: IEEE COMPUTER SOCIETY INTERNATIONAL WORKSHOP ON CLUSTER COMPUTING, IWCC, 1., 1999. **Proceedings...** [S.l.]: IEEE Computer Society, 1999. p.261.

MAILLET, E.; TRON, C. On efficiently implementing global time for performance evaluation on multiprocessor systems. **Parallel Distributed Computing**, Orlando, FL, USA, v.28, n.1, p.84–93, 1995.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: design, implementation, and experience. **Parallel Computing**, [S.l.], v.30, n.5-6, p.817–840, 2004.

MEUER, H.; STROHMAIER, E.; DONGARRA, J.; SIMON, H. D. **Top500 SuperComputer**. 24. ed. 2004. Disponível em: <http://www.top500.org>. Acesso em: 25 Jan. 2005.

MILLER, B. P.; CALLAGHAN, M. D.; CARGILLE, J. M.; HOLLINGSWORTH, J. K.; IRVIN, R. B.; KARAVANIC, K. L.; KUNCHITHAPADAM, K.; NEWHALL, T. The Paradyn Parallel Performance Measurement Tool. **IEEE Computer**, Los Alamitos, CA, USA, v.28, n.11, p.37–46, 1995.

NAGEL, W. E.; ARNOLD, A.; WEBER, M.; HOPPE, H. C.; SOLCHENBACH, K. VAMPIR: visualization and analysis of MPI resources. **Supercomputer**, Netherlands, v.12, n.1, p.69–80, 1996.

NEMETH, Z.; GOMBAS, G.; BALATON, Z. Performance Evaluation on Grids: directions, issues and open problems. In: EUROMICRO CONFERENCE ON PARALLEL, DISTRIBUTED AND NETWORK-BASED PROCESSING, 12., 2004. **Proceedings...** [S.l.]: IEEE Press, 2004. p.290–297.

NETTO, M. A. S.; ROSE, C. D. CRONO: a configurable and easy to maintain resource manager optimized for small and mid-size gnu/linux cluster. In: THE INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 2003, Kaohsiung, Taiwan. **Proceedings...** [S.l.]: IEEE Computer Society Press, 2003. p.555.

NEVES, M. V.; SCHEID, T.; SCHNORR, L. M.; CHARAO, A. S. Integração de Ganglia, libRastro e Pajé para o Monitoramento de Aplicações Paralelas. In: WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 5., 2004, Foz do Iguaçu, Brasil. **Anais...** [S.l.]: Sociedade Brasileira de Computação, 2004.

OTTOGALLI, F.-G.; LABBÉ, C.; OLIVE, V.; STEIN, B.; KERGOMMEAUX, J. C. de; VINCENT, J.-M. Visualisation of Distributed Applications for Performance Debugging. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE-PART II, ICCS, 2001. **Proceedings...** [S.l.]: Springer-Verlag, 2001. p.831–840.

PILLET, V.; LABARTA, J.; CORTES, T.; GIRONA, S. PARAVER: A tool to visualise and analyze parallel code. In: TRANSPUTER AND OCCAM DEVELOPMENTS, WOTUG-18., 1995, Amsterdam. **Proceedings...** [S.l.]: IOS Press, 1995. p.17–31. (Transputer and Occam Engineering, v.44).

REED, D. A. Performance Instrumentation Techniques for Parallel Systems. In: PERFORMANCE EVALUATION OF COMPUTER AND COMMUNICATION SYSTEMS, JOINT TUTORIAL PAPERS OF PERFORMANCE; SIGMETRICS, 1993. **Proceedings...** [S.l.]: Springer-Verlag, 1993. p.463–490.

ROSE, L. D.; ZHANG, Y.; REED, D. A. SvPablo: a multi-language performance analysis system. In: INTERNATIONAL CONFERENCE ON MODELLING TECHNIQUES AND TOOLS FOR COMPUTER PERFORMANCE EVALUATION, 10., 1998, Palma de Mallorca, ES. **Proceedings...** Berlin: Springer-Verlag, 1998. p.352–355. (Lecture Notes in Computer Science, v. 1469).

RUSSELL, R.; CHAVAN, M. Fast Kernel Tracing: a performance evaluation tool for linux. In: IASTED INTERNATIONAL CONFERENCE ON APPLIED INFORMATICS, 19., 2002, Innsbruck, Austria. **Proceedings...** [S.l.]: ACTA Press, 2002.

SACERDOTI, F. D.; MASSIE, M. L.; CULLER, D. E. Wide area cluster monitoring with Ganglia. In: CLUSTER COMPUTING, 2003, Hong Kong. **Proceedings...** [S.l.]: IEEE Press, 2003. p.289–298.

SCHNORR, L. M.; STEIN, B.; NAVAUX, P. Visualização da biblioteca de comunicação DECK em Pajé. In: WORKSHOP DE PROGRAMAÇÃO PARALELA E DISTRIBUÍDA, 2., 2004, Porto Alegre, Brasil. **Anais...** Porto Alegre: GPPD, 2004. p.153–158.

SGI. **Performance Co-Pilot Users and Administrators Guide**. [S.l.: s.n.], 1999.

SILVA, G. J. da; OLIVEIRA STEIN, B. de. Uma biblioteca genérica de geração de rastros de execução para visualização de programas. In: SIMPÓSIO DE INFORMÁTICA DA REGIÃO CENTRO, 2002, Santa Maria, Brasil. **Anais...** [S.l.: s.n.], 2002. v.1.

SILVA, G. J. da; SCHNORR, L. M.; STEIN, B. JRastro: a trace agent for debugging multithreaded and distributed java programs. In: SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 15., 2003. **Proceedings...** Los Alamitos: IEEE Computer Society, 2003. p.46–54.

SOTTILE, M. J.; MINNICH, R. G. Supermon: a high-speed cluster monitoring system. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING, CLUSTER, 2002. **Proceedings...** Los Alamitos: IEEE Computer Society, 2002. p.39.

STEIGNER, C.; WILKE, J. Performance Tuning of Distributed Applications with CoSMoS. In: INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, ICDCS, 21., 2001, Los Alamitos, CA. **Proceedings...** Los Alamitos: IEEE Computer Society, 2001. p.173–180.

STEIN, B.; KERGOMMEAUX, J. C. de; BERNARD, P.-E. Pajé, an interactive visualization tool for tuning multi-threaded parallel applications. **Parallel Computing**, Netherlands, v.26, p.1253–1274, 2000.

TIERNEY, B.; AYDT, R.; GUNTER, D.; SMITH, W.; TAYLOR, V.; WOLSKI, R.; SWANY, M. **A Grid Monitoring Architecture**. [S.l.]: Global Grid Forum, 2002. (Technical Report GWD-PERF-16-3). Disponível em: <http://www-didc.lbl.gov/GGF-PERF/GMA-WG/>. Acesso em: 25 Jan. 2005.

UTHAYOPAS, P.; PHATANAPHEROM, S. Fast and Scalable Real-Time Monitoring System for Beowulf Clusters. In: EUROPEAN PVM/MPI USERS' GROUP MEETING ON RECENT ADVANCES IN PARALLEL VIRTUAL MACHINE AND MESSAGE PASSING INTERFACE, 8., 2001. **Proceedings...** Heidelberg: Springer-Verlag, 2001. p.201. (Lecture Notes in Computer Science, v. 2131).

WALKER, D. W.; DONGARRA, J. J. MPI: a standard Message Passing Interface. **Supercomputer**, [S.l.], v.12, n.1, p.56–68, Jan. 1996.

WIEDENHOFT, G. R.; STEIN, B. Rastreamento e visualização de programas Java usando JVMTI. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 5., 2005, Canoas, Brasil. **Anais...** Canoas: Sociedade Brasileira de Computação, 2005.

YAGHMOUR, K.; DAGENAIS, M. R. Measuring and Characterizing System Behavior Using Kernel-Level Event Logging. In: USENIX ANNUAL TECHNICAL CONFERENCE, USENIX, 2000, Berkeley, CA. **Proceedings...** [S.l.]: USENIX Association, 2000. p.13–26.

APÊNDICE A FERRAMENTAS

A.1 DECK

DECK (BARRETO; RIVIERE; NAVAUX, 1998) é um ambiente de programação paralela para *cluster*. Esse ambiente é composto por uma biblioteca com funções que podem ser utilizadas para o desenvolvimento de aplicações paralelas e por ferramentas para a execução dessas aplicações. Nesta seção serão explicadas a biblioteca DECK e as ferramentas do ambiente de execução.

Biblioteca

O conjunto de funções da biblioteca do ambiente de programação DECK permite que o programador tenha como lidar com:

- múltiplos fluxos de execução
- semáforos, *mutexes* e variáveis condicionais
- comunicação ponto-a-ponto através da caixas de correio
- comunicação coletiva (CASSALI et al., 2000)

As funções que lidam com múltiplos fluxos de execução permitem a criação, finalização, suspensão e sincronização desses fluxos. A implementação dessas funções utiliza a biblioteca `pthreads`, que é largamente utilizada. As funções que lidam com semáforos, *mutexes* e variáveis condicionais são utilizadas quando se programa com múltiplos de execução e visão principalmente evitar a execução de código crítico por mais de um fluxo de execução.

A comunicação no DECK é feita através da abstração de caixas de correio. Um processo que deseja receber dados deve criar uma caixa de correio. Quando outros processos querem enviar dados para esse primeiro processo, eles devem clonar aquela caixa de correio e então colocar nela a mensagem a ser enviada. Com essa abstração, para dois processos se comunicarem devem ser criadas duas caixas de correio, uma por cada processo.

A comunicação coletiva da biblioteca DECK é implementada com as caixas de correio da comunicação ponto-a-ponto. Através das funções coletivas, é possível realizar entre os processos funções de *broadcast*, sincronização, redução, distribuição entre outras.

Execução

O ambiente DECK é composto por um conjunto de ferramentas que realizam a execução das aplicações desenvolvidas com a biblioteca. A ferramenta `deckrun` é responsável

por distribuir as instâncias da aplicação paralela em um conjunto de máquinas. As máquinas que serão utilizadas assim como a quantidade de instâncias a serem executadas são passadas como parâmetro para essa ferramenta.

A.2 Ganglia

Ganglia (MASSIE; CHUN; CULLER, 2004) é uma ferramenta de monitoramento de *cluster*. Ela é composta principalmente por dois programas, `gmond` e `gmetad`, que conjuntamente realizam a coleta e o registro dos dados monitorados, respectivamente. Esses dados monitorados podem ser compostos por informações de utilização de processador, memória, disco e placa de rede, por exemplo. Suas principais características são a flexibilidade e extensibilidade em relação aos dados monitoradas e a sua arquitetura escalável. Nesta seção serão descritos os componentes, a arquitetura e a forma de funcionamento geral da ferramenta.

Componentes

O Ganglia é composto principalmente por duas programas: `gmond` e `gmetad`. Ambos podem ser executados nas máquinas como serviços permanentes.

O programa `gmond` é executado em cada máquina que se deseja monitorar. Ele realiza a coleta de um determinado conjunto de métricas e mantém os dados coletados em memória. Através de uma ferramenta auxiliar, chamada `gmetric`, o conjunto de métricas monitoradas pode ser ampliado. As diferentes execuções do `gmond` se comunicam periodicamente e trocam as informações coletadas localmente. Assim, todas as instâncias do `gmond` tem dados que foram coletados por ele e pelo outros programas nas outras máquinas monitoradas.

O outro programa, `gmetad`, é responsável registrar em arquivo as informações coletadas pelas diferentes execuções do `gmond`. Ele faz isso se conectando a um determinado `gmond`. Como esse `gmond` tem dados de todas as máquinas monitoradas do *cluster*, o `gmetad` apenas obtém as informações e as registra em uma base de dados circular. A forma como foi construído esse programa permite que ele se conecte com outros `gmetad` para obter informações que ele obteria de um `gmond`. Isso flexibiliza a arquitetura da ferramenta Ganglia permitindo que seja escalável.

Arquitetura

Os dois principais programas do Ganglia, `gmond` e `gmetad`, podem ser organizados hierarquicamente. A Figura A.1 mostra essa organização onde existem conjuntos de máquinas monitoradas pela combinação de um conjunto de `gmond`'s e um `gmetad` responsável. Esses conjuntos de máquinas podem ser tanto *clusters* independentes quanto parte de um *cluster*maior.

Funcionamento

Cada `gmond` é responsável pelo monitoramento de uma máquina. Ele realiza esse monitoramento através de um conjunto de *scripts*, sendo que cada *script* é responsável por coletar determinada informação. Novos scripts podem ser adicionados permitindo que novas métricas possam ser coletadas através da organização arquitetural do Ganglia.

Há um canal de comunicação *multicast* que permite que os `gmond`'s compartilhem as informações por eles coletadas. A Figura A.2 ilustra esse canal de comunicação. Periodi-

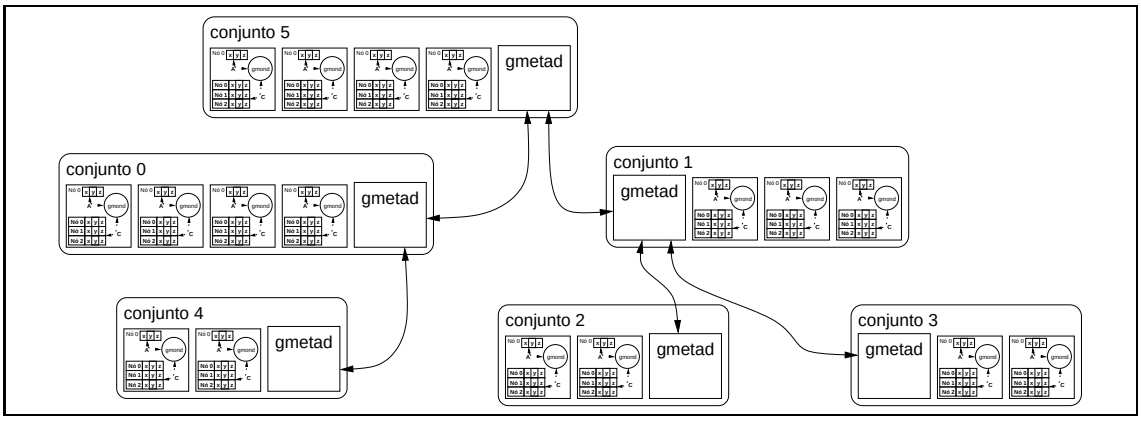

Figura A.1: Arquitetura hierárquica da ferramenta Ganglia utilizando a combinação de suas duas principais ferramentas, `gmond` e `gmetad`

camente e de forma independente, cada `gmond` envia através desse canal todas as informações locais coletadas por ele. Os outros `gmond`'s que estiverem participando desse canal multicast receberão as informações e as manterão em memória. Os intervalos periódicos de comunicação e a criação de múltiplos canais multicast separando hierarquicamente os monitoradores `gmond` diminuem a intrusão do monitoramento e das trocas de dados.

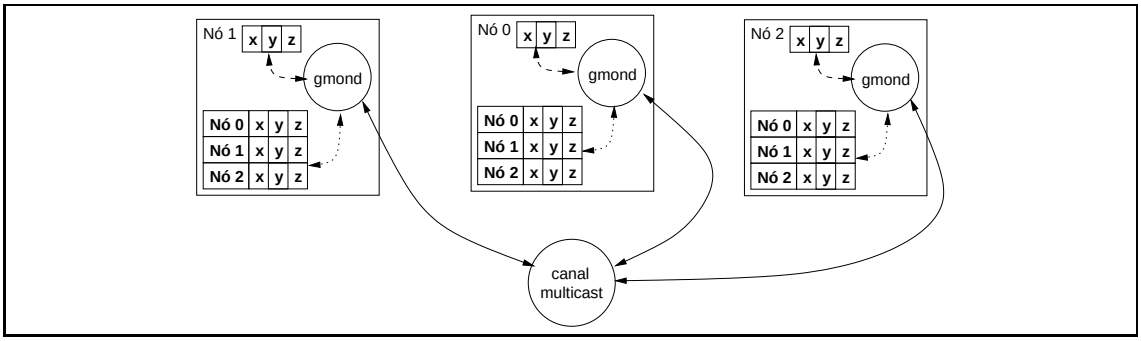

Figura A.2: Forma de funcionamento da troca de informações entre as instâncias dos `gmond`'s através do canal multicast do Ganglia

Além do canal de comunicação multicast, cada `gmond` mantém um outro ponto de conexão. Através desse ponto de conexão, o `gmetad` ou uma aplicação externa podem obter as informações mantidas em memória pelo `gmond`.

A sincronização dos dados coletados pelas diferentes instâncias dos `gmond`'s acontece de acordo com o relógio da máquina onde é executado o `gmond` usado para obter as informações. Por exemplo, caso existam dois nós monitorados, cada um com seu respectivo `gmond`. Quando o `gmetad` ou outra aplicação for obter as informações desses `gmond`'s, essas serão sincronizadas de acordo com o relógio da máquina onde foram coletados os dados.

A.3 Performance Co-Pilot

O Performance Co-Pilot (PCP) (SGI, 1999) é uma ferramenta de monitoramento de *cluster*. Essa ferramenta é composta por agentes de monitoramento, uma biblioteca para

construção de novos módulos para esses agentes e ferramentas auxiliares. Nesta seção serão detalhados os componentes, a arquitetura e a forma de funcionamento do PCP.

Componentes

Os principais componentes do Performance Co-Pilot são: `pmcd` e `pmlogger`. Através desses dois componentes é possível realizar o monitoramento em uma máquina. O `pmcd` é composto por um conjunto de módulos, chamados `pmda`'s. Cada `pmda` é responsável por monitorar dados específicos. Por exemplo, existe um `pmda` para monitorar as informações relacionadas a memória, outro para coletar dados sobre o sistema de arquivos e assim por diante. A responsabilidade do `pmcd` é manter as informações coletadas pelo `pmda` em memória. O `pmcd` não mantém nenhum tipo de registro histórico das informações, de forma que o dado que está em memória é sempre o último que foi coletado por determinado `pmda`.

O `pmlogger` é quem registra as informações coletadas por um ou mais `pmcd`'s. Esse registro começa com a obtenção das informações através de uma determinada porta do `pmcd` e depois na gravação de um arquivo com um formato específico do PCP. Através da biblioteca disponibilizada junto com o Performance Co-Pilot é possível acessar as informações registradas nesse arquivo.

Arquitetura

As duas ferramentas, `pmcd` e `pmlogger`, podem ser combinadas de diferentes formas para compor a arquitetura do PCP. Cada máquina monitorada deve ter um `pmcd` associado. Esse programa fica então responsável por coletar as informações locais daquela máquina. Nesta seção são mostradas duas alternativas arquiteturais na combinação dessas duas ferramentas.

A execução do `pmlogger` em cada máquina que contém um `pmcd` é uma forma de combinar as duas ferramentas. Através dessa combinação, o tráfego de dados na rede que interconecta o *cluster* é evitado, mas cada máquina contém dois serviços que são executados concorrentemente com as aplicações normais, aumentando a intrusão.

Outra forma de combinar as ferramentas é associar um `pmlogger` a um conjunto de `pmcd`'s. Essa alternativa aumenta a utilização da rede de interconexão do *cluster*, mas diminui a intrusão em cada máquina e permite a escalabilidade da ferramenta.

Funcionamento

A forma de funcionamento do PCP, independente da combinação arquitetural adotada, é uma interação entre os componentes `pmlogger` e `pmcd` e seus módulos `pmda`. A Figura A.3 mostra a interação entre esses componentes durante o funcionamento do Performance Co-Pilot. A flecha 1 indica a inicialização do `pmlogger` através de um arquivo de configuração. Nesse arquivo, fica estabelecido quais os dados serão coletados e a periodicidade dessa coleta. O PCP permite que um conjunto de dados seja coletado com um determinado período e outro conjunto com outro período. A flecha 2 representa a consulta do `pmlogger` ao `pmcd` de acordo com as configurações passadas anteriormente. O resultado dessa consulta é passada ao `pmlogger` na flecha 3 e então os dados são gravados em arquivo, na flecha 4. Paralelo ao funcionamento do `pmlogger`, o `pmcd` fica se comunicando com seus módulos `pmda` através das flechas A e B de acordo com a alteração dos dados monitorados pelo `pmda`.

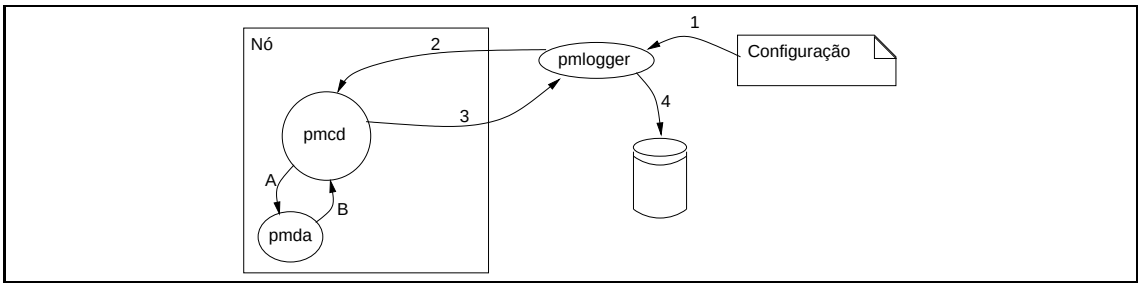

Figura A.3: Funcionamento do Performance Co-Pilot mostrando a interação entre o `pmlogger`, `pmcd` e o `pmda`

A.4 libRastro

A biblioteca libRastro (SILVA; OLIVEIRA STEIN, 2002) é composta por um conjunto de funções genéricas utilizadas para instrumentar aplicações. Suas principais características são o suporte a múltiplos fluxos de execução, baixa intrusividade, sincronização de relógios e funções estensíveis às necessidades do programador. Sua utilização já foi validada na construção de um rastreador de aplicações Java (SILVA; SCHNORR; STEIN, 2003). Nesta seção será explicada a forma de utilização da biblioteca e o seu funcionamento.

Utilização

A técnica de instrumentação (REED, 1993) utilizada pela libRastro consiste em adicionar instruções dentro de um código já existente de um programa. A libRastro oferece ao programador um conjunto de funções genéricas que podem ser utilizadas para instrumentar programas. Essas funções realizam o registro de eventos e no final oferecem ao programar um arquivo binário com os eventos gravados, além de funções que podem ser utilizadas para ler desse arquivo binário.

Além das funções normais oferecidas pela biblioteca libRastro, o programador pode utilizar um conjunto de funções padronizadas de acordo com as suas necessidades. Estas funções, utilizadas pelo programador com nomes pré-estabelecidos e parâmetros específicos, têm seus códigos gerados automaticamente por uma aplicação que varre os códigos da aplicação instrumentada e gera um arquivo com a implementação das funções. Esse arquivo com as implementações deve então ser compilado juntamente com a aplicação instrumentada.

A libRastro oferece uma ferramenta de sincronização caso o programador necessite sincronizar os eventos registrados em diferentes máquinas com diferentes relógios. Essa ferramenta registra as diferenças de relógio entre uma máquina de referência e as máquinas onde são registrados os eventos e gera um arquivo. O registro das diferenças deve ser feito antes e depois da execução da aplicação instrumentada. A sincronização, utilizando a técnica proposta por (MAILLET; TRON, 1995), é feita no momento da leitura dos arquivos com os eventos registrados.

Funcionamento

A libRastro mantém um *buffer* em memória onde os eventos vão sendo registrados quando a aplicação instrumentada é executada. Esse *buffer* tem um tamanho fixo e quando fica cheio, os eventos são registrados em arquivo. Existe um *buffer* e um arquivo para cada

fluxo de execução da aplicação instrumentada.

O resultado da execução de uma aplicação instrumentada com a libRastro é um conjunto de arquivos. Cada arquivo é composto pelos eventos que foram registrados durante a execução dessa aplicação. O programador pode analisar os eventos através de uma ferramenta da libRastro ou utilizar a própria biblioteca para converter o formato dos dados e então visualizá-los em alguma ferramenta de visualização.

A.5 Pajé

Pajé (STEIN; KERGOMMEAUX; BERNARD, 2000) é uma ferramenta genérica de visualização de informações. Sua adaptabilidade permite que seja usada para visualizar desde o comportamento de aplicações paralelas até gráficos quantitativos de dados de monitoramento de *cluster*. Nesta seção serão descritas as principais características da ferramenta e o formato do arquivo de entrada do Pajé.

Principais características

As principais características do Pajé são interatividade, escalabilidade e extensibilidade. A interatividade do Pajé advém do fato de que, através de sua interface gráfica, é possível inspecionar os objetos que estão sendo visualizados. Esta inspeção permite que o programador tenha dados mais detalhados acerca desses objetos. A escalabilidade do Pajé é possível porque através da ferramenta é possível visualizar um conjunto grande de objetos. A extensibilidade acontece porque Pajé foi construído utilizando componentes. Novos componentes podem ser adicionados a implementação podendo dar novas funcionalidades de acordo com a evolução das necessidades de visualização.

Formato de arquivo

Pajé define um formato de arquivo. Esse formato de arquivo trabalha com eventos sendo que existem eventos para a construção de objetos visuais, e não para a construção de objetos de acordo com a semântica, por exemplo, de uma aplicação paralela. Por exemplo, não há eventos que definem uma comunicação, mas sim eventos que dizem para iniciar uma flecha em determinada data e outro evento para finalizar esta flecha. Esta representação gráfica pode ser usada para visualizar uma comunicação quando se está tratando da visualização de uma aplicação paralela.

Essa característica de não atuar junto a semântica de um ambiente de programação permite ao Pajé ser facilmente utilizada para qualquer tipo de informação. Para tal, deve-se primeiro definir a hierarquia das informações que serão utilizadas e então utilizar eventos específicos para criar objetos dentro dessa hierarquia.

APÊNDICE B PROTÓTIPO DE INTEGRAÇÃO

O protótipo de integração foi construído com o objetivo de implementar as etapas do processo de unificação de dados explicado no capítulo 3. Na seção 4.2 do capítulo 4 foram explicadas as decisões de projeto do protótipo.

Neste apêndice, serão vistos a forma de estender o protótipo para que seja possível adicionar novas fontes de informação ao processo de integração. Depois, um manual de utilização desse protótipo é apresentado.

B.1 Adição de novos módulos

A utilização da orientação a objetos na construção do protótipo permite que novas funcionalidades sejam adicionadas. Essas funcionalidades são normalmente novas fontes de informação que podem ser integradas com os dados já existentes.

Os requisitos para a utilização de novas fontes de informação estão delineadas no capítulo 3. Nesta seção serão vistas quais as classes que devem ser implementadas para se adicionar uma nova fonte de dados no protótipo.

A adição de uma nova fonte de informação ao protótipo deve começar através da criação de sub-classes para as classes `FileReader` e `Converter`. A sub-classe de `FileReader` a ser implementada deve retornar eventos do tipo da nova fonte de dados. Deve-se então construir uma sub-classe para a classe `Event`. Para inicializar a nova fonte de dados deve-se adicionar um método à classe `DataSource` de forma que seja possível a inicialização correta do componente. Esse método construído deve ser chamado de acordo com a descrição das informações passadas a classe `DataControl`, de forma que o método de inicialização dessa classe deve ser alterado.

A implementação da nova sub-classe de `FileReader` está atrelada ao formato do arquivo de rastreamento gerado pela nova fonte de informação. Essa nova sub-classe deve retornar, quando chamado o método de leitura de eventos, eventos da nova sub-classe de `Event` construída.

A implementação da nova sub-classe de `Converter` deve levar em conta o processo de padronização realizado para as fontes de informação já existentes. Esse processo consiste em definir a forma de visualização individual para os dados dessa fonte de informação e a forma como essa visualização ficará integrada com as outras informações. Esse processo consiste também em definir a hierarquia de entidades da nova fonte de informação, processo também descrito no capítulo 4.2.

Exemplo - Adição hipotética de uma fonte de dados PVM

Supondo que exista uma forma de se rastrear aplicações desenvolvidas com a biblioteca PVM de passagem de mensagem. O resultado hipotético desse rastreamento é um arquivo que contém os principais eventos da execução da aplicação.

O primeiro passo na adição dessa nova fonte de dados é descobrir a hierarquia e entidades do PVM e planejar a integração dos objetos dessa entidade com as informações oriundas de outras fontes de dados, como uma ferramenta de monitoramento. Depois disso deve ser definida a forma de visualização das informações rastreadas do PVM.

O próximo passo é a construção do componente fonte de dados para PVM. Nesse passo, as classes `PVMFileReader`, `PVMConverter` e `PVMEvent` devem ser implementadas de acordo com as decisões tomadas no primeiro passo referentes a padronização dos dados e a definição hierárquica. A classe `PVMFileReader` deve realizar a sincronização dos eventos cooperando com objetos da classe `Sync` e implementar os métodos da classe abstrata `FileReader`. A classe `PVMConverter` deve implementar os métodos da classe abstrata `Converter` e realizar a padronização das informações utilizando a definição hierárquica definida pelas classes `Conversion` e `DataControl`.

No fim, deve ser criado um método `initToPVM` na classe `DataSource` e alterar o método da classe `DataControl` que realiza a inicialização das múltiplas fontes de informações de acordo com uma descrição de quais dados devem ser integrados.

B.2 Manual de utilização

Este manual pretende mostrar aos usuários a forma de utilização da ferramenta protótipo de integração criada. A seguir, são listados os parâmetros que podem ser passados ao protótipo e depois alguns exemplos de utilização.

Parâmetros do protótipo

A Tabela B.1 mostra os parâmetros aceitos pelo protótipo. Os colchetes representam opções facultativas e as chaves opções obrigatórias.

Estes parâmetros podem ser estendidos caso novos componentes de fontes de dados sejam adicionados ao protótipo.

Tabela B.1: Parâmetros aceitos pelo protótipo de integração construído

`[-deck {deck1.rst, ..., deckN.rst}]` onde `deck1.rst, ..., deckN.rst` são arquivos de rastreamento de uma aplicação desenvolvida com a biblioteca DECK
`[-mpi {mpi.clog}]` onde `mpi.clog` é o arquivo de rastreamento gerado pela biblioteca MPE na execução de uma aplicação MPI
`[-mpinodes {mpi.nodes}]` onde `mpi.nodes` é o arquivo que contém os nomes das máquinas utilizadas para a execução da aplicação MPI. Opção obrigatória caso se use o parâmetro `-mpi`
`[-os {machine1.os, ...,machineN.os}]` onde `machine1.os, ...,machineN.os` são os arquivos de rastreamento do sistema operacional Linux
`[-ganglia {ganglia.traced}]` onde `ganglia.traced` é o arquivo que contém o registro das informações que foram coletadas pela ferramenta de monitoramento Ganglia
`[-pcp {pcp1, ..., pcpN}]` onde `pcp1, ..., pcpN` são os arquivos gerados pela ferramenta de monitoramento de *cluster* Performance Co-Pilot
`[-sync {timesync}]` onde `timesync` é o arquivo que contém as diferenças de relógio entre as máquinas monitoradas e uma máquina de referência
`{-hierarchy {hierarchy.plist}}` onde `hierarchy.plist` é o arquivo que contém as definições hierárquicas que devem ser utilizadas na integração
`{-output {output.trace}}` onde `output.trace` é o arquivo de saída que terá as informações integradas e que pode ser utilizada para visualizar com a ferramenta Pajé

Exemplos de utilização

Nesta seção serão mostrados e comentados alguns exemplos de utilização do protótipo com diferentes tipos de integração de dados.

1. Integração de informações das ferramentas de monitoramento de *cluster* Ganglia e Performance Co-Pilot:
```
./Integrador.app/Integrador \
-sync timesync \
-hierarchy ganglia+pcp.plist \
-ganglia ganglia.traced \
-pcp pcp_* \
-output ganglia+pcp.trace
```

2. Integração de informações da ferramenta de monitoramento Ganglia, dados do sistema operacional e rastreamento de uma aplicação MPI:
```
./Integrador.app/Integrador \
-sync timesync \
```

```
-hierarchy ganglia+os+mpi.plist \
-ganglia ganglia.traced \
-os *.os \
-mpi a.out.clog \
-output ganglia+os+mpi.trace
```

3. Conversão simples de informações da ferramenta Ganglia para o formato do Pajé (nesse caso o arquivo de sincronização não foi utilizado pois o Ganglia já realiza a sincronização dos eventos):
```
./Integrador.app/Integrador \
-hierarchy ganglia.plist \
-ganglia ganglia.traced \
-output ganglia.trace
```

4. Integração de informações de rastreamento de uma aplicação DECK, do sistema operacional Linux, do Ganglia e do Performance Co-Pilot:
```
./Integrador.app/Integrador \
-sync timesync \
-hierarchy deck+os+ganglia+pcp.plist \
-deck *.rst \
-os *.os \
-ganglia ganglia.traced \
-pcp pcp_* \
-output deck+os+ganglia+pcp.trace
```

5. Integração de informações do rastreamento de uma aplicação MPI com dados do Performance Co-Pilot:
```
./Integrador.app/Integrador \
-sync timesync \
-hierarchy mpi+pcp.plist \
-mpi a.out.clog \
-pcp pcp_* \
-output mpi+pcp.trace
```

APÊNDICE C MAPEAMENTO DE EVENTOS

Este anexo tem por objetivo mostrar todos os eventos registrados nas fontes de informação utilizadas no trabalho e seus respectivos mapeamentos. Nesta seção são mostrados os mapeamentos realizados na conversão de dados do DECK, do MPI, do Ganglia, dos dados registrados no Linux e finalmente dos dados coletados pelo Performance Co-Pilot.

C.1 DECK

Esta seção apresenta os mapeamentos de eventos gerados na biblioteca DECK para eventos Pajé, de forma que atinja a visualização desejada descrita na seção 4.2. A Tabela C.1 mostra os mapeamentos dos eventos globais gerados no DECK, responsáveis por rastrear a criação e destruição de processos. A primeira coluna desta tabela mostra o código do evento, a segunda o nome do evento e a terceira o conjunto de eventos correspondentes em Pajé utilizados.

Tabela C.1: Mapeamentos dos eventos globais gerados na biblioteca DECK para eventos Pajé. Esse mapeamento foi utilizado no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
101	`deck_init_node`	`PajeCreateContainer`,`PajeSetState`
102	`deck_finalize_node`	-

Tabela C.2: Mapeamentos dos eventos gerados na comunicação por caixa de mensagens da biblioteca DECK para eventos Pajé. Esse mapeamento foi utilizado no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
401	`deck_mbox_create`	`PajeNewEvent`
402	`deck_mbox_clone_in`	`PajeSetState`
403	`deck_mbox_clone_out`	`PajeSetState`,`PajeStartLink`,`PajeEndLink`
404	`deck_mbox_destroy`	`PajeNewEvent`
405	`deck_mbox_post`	`PajeNewEvent`,`PajeNewEvent`,`PajeStartLink`
406	`deck_mbox_retrv_in`	`PajeSetState`
407	`deck_mbox_retrv_out`	`PajeSetState`,`PajeEndLink`

A Tabela C.2 apresenta os mapeamentos feitos na conversão dos eventos gerados na parte de comunicação por caixa de mensagens do DECK. A primeira coluna tem os iden-

tificadores, a segunda o nome dos eventos gerados e a terceira os eventos correspondentes em Pajé utilizados.

A Tabela C.3 mostra os mapeamentos utilizados na conversão dos eventos de gerenciamento de fluxos de execução da biblioteca DECK. Esses eventos compreendem desde a criação e destruição dos fluxos de execução, mas também das funções que trocam o estado do fluxo de execução, como barreiras de sincronização e ordens explícitas para suspender a execução do fluxo. A primeira coluna dessa tabela mostra os identificadores, a segunda o nome dos eventos e a terceira os eventos Pajé correspondentes.

Tabela C.3: Mapeamentos dos eventos gerados no gerenciamento de fluxos de execução da biblioteca DECK para eventos Pajé. Esse mapeamento foi utilizado no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
201	`deck_thread_create`	`PajeStartLink`
202	`deck_thread_created`	`PajeSetState`,`PajeEndLink`
203	`deck_thread_join_in`	`PajeSetState`
204	`deck_thread_join_out`	`PajeStartLink`,`PajeEndLink`
205	`deck_thread_yield_in`	`PajeSetState`
206	`deck_thread_yield_out`	`PajeSetState`
207	`deck_thread_sleep_in`	`PajeSetState`
208	`deck_thread_sleep_out`	`PajeSetState`
209	`deck_thread_usleep_in`	`PajeSetState`
210	`deck_thread_usleep_out`	`PajeSetState`
211	`deck_thread_barrier_in`	`PajeSetState`
212	`deck_thread_barrier_out`	`PajeSetState`
213	`deck_thread_done`	-

A Tabela C.4 mostra os eventos gerados na parte de comunicação coletiva do DECK e seus respectivos mapeamentos para eventos Pajé. A primeira coluna dessa tabela contém os identificadores, a segunda tem o nome dos eventos e a terceira contém os eventos correspondentes em Pajé para cada evento registrado no DECK. Os eventos com identificação 326, 328, 330 e 334 não tem eventos correspondentes em Pajé, mas seus registros contêm informações que são necessárias para a conversão de outros eventos.

Todos os mapeamentos descritos nesta seção foram utilizados no componente conversor DECK do protótipo apresentado no capítulo 4. Esse conversor faz parte da padronização dos dados proposta no modelo apresentado no capítulo 3.

Tabela C.4: Mapeamentos dos eventos gerados nas funções da comunicação coletiva da biblioteca DECK para eventos Pajé. Esse mapeamento foi utilizado no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
301	deck_group_create_in	PajeNewEvent
302	deck_group_destroy	PajeNewEvent
303	deck_group_barrier_in	PajeSetState
304	deck_group_barrier_out	PajeSetState
305	deck_group_bcast_in	PajeSetState
306	deck_group_bcast_out	PajeSetState
307	deck_group_reduce_in	PajeSetState
308	deck_group_reduce_out	PajeSetState
309	deck_group_msgbcast_in	PajeSetState
310	deck_group_msgbcast_out	PajeSetState
311	deck_group_scatter_in	PajeSetState
312	deck_group_scatter_out	PajeSetState
313	deck_group_gather_in	PajeSetState
314	deck_group_gather_out	PajeSetState
315	deck_group_allgather_in	PajeSetState
316	deck_group_allgather_out	PajeSetState
317	deck_group_allreduce_in	PajeSetState
318	deck_group_allreduce_out	PajeSetState
319	deck_group_create_out	PajeSetState
320	deck_group_bcast_send	PajeStartLink * N
321	deck_group_bcast_recv	PajeEndLink
322	deck_group_msgbcast_send	PajeStartLink * N
323	deck_group_msgbcast_recv	PajeEndLink
324	deck_group_scatter_send	PajeStartLink * N
325	deck_group_scatter_recv	PajeEndLink
326	deck_group_gather_send	Salva informações do evento para serem usadas em deck_group_gather_recv
327	deck_group_gather_recv	PajeStartLink * N, PajeEndLink * N
328	deck_group_reduce_send	Salva informações do evento para serem usadas em deck_group_reduce_recv
329	deck_group_reduce_recv	PajeStartLink * N, PajeEndLink * N
330	deck_group_allgather_gsend	Salva informações do evento para serem usadas em deck_group_allgather_grecv
331	deck_group_allgather_grecv	PajeStartLink * N, PajeEndLink * N
332	deck_group_allgather_bsend	PajeStartLink * N
333	deck_group_allgather_brecv	PajeEndLink
334	deck_group_allreduce_rsend	Salva informações do evento para serem usadas em deck_group_allreduce_rrecv
335	deck_group_allreduce_rrecv	PajeStartLink * N, PajeEndLink * N
336	deck_group_allreduce_bsend	PajeStartLink * N
337	deck_group_allreduce_brecv	PajeEndLink

C.2 MPI

Esta seção apresenta os mapeamentos utilizados no trabalho e que são utilizados para se converter eventos de uma aplicação MPI em eventos Pajé. A Tabela C.5 mostra os mapeamentos desses eventos MPI. A primeira coluna da tabela mostra o código do evento, seguido na coluna pelo nome dos eventos e na terceira coluna pelos eventos correspondentes no formato do Pajé.

Tabela C.5: Mapeamentos dos eventos gerados no rastreamento de aplicações MPI para eventos Pajé. Esse mapeamento foi feito no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
155	`mpi_receive_in`	`PajeSetState`
156	`mpi_receive_out`	`PajeSetState`
161	`mpi_send_in`	`PajeSetState`
162	`mpi_send_out`	`PajeSetState`
-101	`mpi_send`	`PajeStartLink`
-102	`mpi_recv`	`PajeEndLink`
4	`mpi_begin`	`PajeSetState`

C.3 Ganglia

Esta seção apresenta os mapeamentos implementados no protótipo e que realizam a conversão dos dados coletados pelo Ganglia para o formato do Pajé. A Tabela C.6 mostra esses mapeamentos.

Tabela C.6: Mapeamentos dos eventos gerados pela ferramenta de monitoramento Ganglia para eventos Pajé. Esse mapeamento foi feito no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
1	`GANGLIA_METRIC`	`PajeDefineVariableType`
2	`GANGLIA_DATA`	`PajeSetVariable`

C.4 Sistema Operacional Linux

Esta seção apresenta os mapeamentos implementados no protótipo e que realizam a conversão dos dados registrados dentro do sistema operacional Linux para o formato do Pajé. A Tabela C.7 mostra esses mapeamentos.

Tabela C.7: Mapeamentos dos eventos gerados dentro do sistema operacional Linux para eventos Pajé. Esse mapeamento foi feito no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
0	`TASK_RUNNING`	`PajeSetState`
1	`TASK_INTERRUPTIBLE`	`PajeSetState`

C.5 Performance Co-Pilot

Esta seção apresenta os mapeamentos implementados no protótipo e que realizam a conversão dos dados coletados pelo Performance Co-Pilot para o formato do Pajé. A Tabela C.8 mostra esses mapeamentos.

Tabela C.8: Mapeamentos dos eventos gerados pela ferramenta de monitoramento Performance Co-Pilot para eventos Pajé. Esse mapeamento foi feito no protótipo descrito no capítulo 4

Id	Tipo do evento	Eventos Pajé correspondentes
-	`PCP_EVENT`	`PajeDefineVariableType`, `PajeSetVariable`