UFSM

Trabalho de Graduação

REESTRUTURAÇÃO DA FERRAMENTA PAJÉ PARA AUMENTO DE ESCALABILIDADE

Edmar Pessoa Araújo Neto

Curso de Ciência da Computação

Santa Maria, RS, Brasil

2005

REESTRUTURAÇÃO DA FERRAMENTA PAJÉ PARA AUMENTO DE ESCALABILIDADE

por

Edmar Pessoa Araújo Neto

Trabalho de Graduação apresentado ao Curso de Ciência da Computação – Bacharelado, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Curso de Ciência da Computação

Trabalho de Graduação nº 202

Santa Maria, RS, Brasil

2005

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada, aprova o Trabalho de Graduação

REESTRUTURAÇÃO DA FERRAMENTA PAJÉ PARA AUMENTO DE ESCALABILIDADE

elaborado por

Edmar Pessoa Araújo Neto

como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Comissão Examinadora:

Prof. Dr. Benhur de Oliveira Stein
(Orientador)

Prof^a. Dr^a. Iara Augustin

Prof^a. Dr^a. Andrea Schwertner Charão

Santa Maria, 16 de Dezembro de 2005.

It is not so very important for a person to learn facts. For that he does not really need a college. He can learn them from books. The value of an education in a college is not the learning of many facts but the training of the mind to think something that cannot be learned from textbooks.

(Albert Einstein)

Dedicado aos meus pais,

Paulo e Kátia.

Obrigado por tanto amor e apoio.

Agradecimentos

Agradeço aos meus pais, Paulo e Kátia, pela força, pelo apoio e por serem sempre um porto seguro que posso recorrer quando me sinto perdido. Agradeço também às minhas irmãs, Luane e Elane, que mesmo sem saber me incentivaram durante a graduação, puxando minha orelha quando eu estava muito relaxado. Agradeço também à minha segunda mãe, Vera Lúcia, por nunca me deixar sozinho e por sempre me ajudar nos momentos onde mais ninguém pode fazer algo por mim. Agradeço à Deus, por ser a inteligência suprema, causa primária de todas as coisas. Eterno, imutável, único, onipotente, soberanamente justo e bom. Quando eu crescer quero ser igual a ele.

Agradeço aos meus colegas, por sempre se manterem unidos, se incentivando e fazendo o possível para que ninguém caísse do barco durante essa viajem de graduação. Um agradecimento também à todos do LSC, em especial ao Veiga, Elton, Geovani, Scheid, Lucas e Diego. Ao Veiga (vulgo Marcelo), pelas conversas viajadas e pela parceria eterna pra tomar cerveja nas brumas do Aldeia. Ao Elton (eterno trovador), pela paciência necessária pra aturar as minhas incessantes perguntas sobre tudo, por me apresentar o *stumble*, pelo incentivo nos momentos de preguiça e pelos conselhos. Ao Geovani, por ser o pilar de responsabilidade do LSC, por ter paciência pra aturar nossas brincadeiras sempre de bom humor e por nunca negar ajuda. Ao Scheid (vulgo Tucunduva), pelo jeito único de ser, por sempre (com ênfase no sempre) ser parceria para as festas (e pro RPG), por se manter sóbrio (muitas vezes o único sóbrio) nas festas e pelas conversas sobre cultura inútil. Ao Lucas, por sempre estar disposto a ajudar e pelos conselhos nos momentos de dúvida. Ao

Alemão, por ser um modelo de produtividade e por sempre acreditar na capacidade do pessoal do LSC.

Agradeço a todos do Curso de Ciência da Computação, em especial aos professores Benhur e Pasin e a mais do que funcionária Nelma. Ao Benhur, pela infinita paciência, por ser um modelo de conhecimento e humildade e pela orientação durante a minha graduação, estando presente na maioria das vezes que precisei de direção (inclusive nos domingos!). Ao Pasin, por ter me convidado a entrar no LSC, fazendo com que eu não desistisse da ciência da computação. À digníssima Nelma, por sempre estar disposta a ajudar e por ser um exemplo de profissional.

Agradeço aos meus amigos por sempre me aturarem, pelas longas conversas, pelas festas, pelos jogos de RPG, pela ajuda nos momentos de dificuldade, por me ouvir reclamar da vida e reclamarem junto comigo, por dar risadas das minhas bobagens e por terem feito com que eu me sentisse em casa aqui em Santa Maria. Um agradecimento especial aos meus irmãos Franciano e Jé, pelos chimarrões no trevo, pelas noitadas de jogatina e pelo grupo de auto-ajuda do CHMMMMMM.

Um agradecimento especial a Pam, por ter sido minha companheira, por ter me amado e por ter marcado minha vida de tantas maneiras que não existem palavras para expressar o que ela representa para mim.

Por fim, gostaria de agradecer ao Google, por ser uma fonte de conhecimento sempre disponível e por ter e estar redefinindo a internet. Se o mundo fosse dominado pelo Google, com certeza ele seria um lugar melhor (*by* Boufleur). E também à Ambev, pela única e inigualável Bohemia, responsável por vários esclarecimentos sobre a vida, o universo e tudo mais.

Sumário

Lista de Tabelas

Lista de Figuras

RESUMO

Trabalho de Graduação
Ciência da Computação
Universidade Federal de Santa Maria

REESTRUTURAÇÃO DA FERRAMENTA PAJÉ PARA AUMENTO DE ESCALABILIDADE

AUTOR: EDMAR PESSOA ARAÚJO NETO

ORIENTADOR: PROF. DR. BENHUR DE OLIVEIRA STEIN

Data e Local da Defesa: Santa Maria, 16 de Dezembro de 2005.

A visualização de aplicações concorrentes permite um melhor entendimento e análise do comportamento assumido pelas mesmas durante sua execução. Existem várias maneiras de se visualizar aplicações, e uma delas é através dos rastros de execução. Este trabalho descreve as alterações feitas na ferramenta de visualização de rastros de execução Pajé. Estas alterações foram implementadas a fim de aumentar a escalabilidade da ferramenta, permitindo a visualização de rastros de aplicações maiores com um melhor desempenho.

Capítulo 1

Introdução

É crescente a criação de aplicações que necessitam de grande poder computacional para resolução de problemas com resultados rápidos. A programação concorrente é utilizada para tentar aumentar o desempenho destas aplicações. Utilizando a programação concorrente, busca-se aumentar o desempenho das aplicações através da divisão de suas tarefas em vários fluxos de execução situados em diferentes processadores e/ou máquinas.

Os ambientes de desenvolvimento para programação concorrente ainda carecem de ferramentas que auxiliem na análise de aplicações concorrentes. Grande parte das ferramentas que auxiliam a programação seqüencial tem poder limitado quando utilizadas neste enfoque.

Pajé [OLI 99, KER 2000, OLI 2000] é uma ferramenta em desenvolvimento no Laboratório de Sistemas de Computação (LSC/UFSM), que permite a visualização do comportamento que um programa concorrente teve durante sua execução e, por isso, é especialmente útil nestes ambientes, pois facilita a análise destas aplicações. Atualmente, Pajé possui alguns problemas de desempenho relacionados à escalabilidade. Escalabilidade neste texto pode ser abordada de duas formas: escalabilidade de visualização, que diz respeito à quantidade excessiva de informações exibidas na tela e escalabilidade de processamento que diz respeito ao processamento necessário a ser feito para exibir as informações requisitadas.

Este trabalho consiste na reestruturação dos módulos de gerenciamento de memória do Pajé, e tem como objetivo aumentar a escalabilidade de visualização e de

processamento desta ferramenta durante a análise do comportamento de aplicações concorrentes.

No capítulo 2 contém uma contextualização deste trabalho, onde é descrito mais detalhadamente a análise de aplicações paralelas e de algumas ferramentas de visualização. O capítulo 3 explica o funcionamento do Pajé, analisando os módulos que compõem a ferramenta. No capítulo 4 é feito um detalhamento da reestruturação dos módulos de gerenciamento de memória do Pajé. O capítulo 5 demonstra os resultados alcançados, onde é feita uma comparação entre a antiga e a nova implementação do Pajé. Por fim, o capítulo 6 apresenta uma conclusão deste trabalho e também cita alguns trabalhos futuros.

Capítulo 2

Visualização de Programas

Algumas aplicações concorrentes são desenvolvidas para alcançar o maior desempenho possível na execução de suas tarefas. Portanto, a depuração destas aplicações vai além da busca de erros e falhas - ela deve também buscar melhorias para que o máximo desempenho possa ser alcançado. Este capítulo aborda a depuração de aplicações concorrentes, detalhando este processo e apresentando algumas ferramentas que auxiliam o desenvolvedor nesta tarefa.

2.1 Depuração de Aplicações Concorrentes

A depuração de aplicações consiste na busca de falhas na execução ou maneiras de melhorar o desempenho. Esta depuração é mais complexa na programação concorrente que na seqüencial pois envolve vários processos que podem possuir um comportamento não determinístico. Além disso, a depuração de aplicações concorrentes geralmente tem um foco maior na busca por um aumento de desempenho para melhorar o tempo de resposta da aplicação.

Existem várias maneiras de se depurar aplicações concorrentes e uma delas é através da visualização da execução da aplicação. Esta visualização pode ser feita a partir de um arquivo de rastros gerado durante sua execução, contendo registros de eventos ocorridos. A visualização descreverá o comportamento da aplicação, mesmo se ela estiver distribuída em vários fluxos de execução. Possibilitando a busca por melhorias, já que se pode analisar os eventos ocorridos e as comunicações entre os

fluxos.

2.2 Rastreamento de Aplicações

Os rastros descrevem o comportamento de uma execução, e através deles o desenvolvedor pode tirar conclusões sobre sua aplicação. Sendo assim, é importante que os rastros contenham informações significativas sobre a aplicação. Porém, deve-se ter cuidado para que somente as informações necessárias sejam registradas, pois os rastros são gerados durante a execução e isso pode interferir na aplicação, podendo até ocultar erros. Quando se deseja rastrear uma aplicação, deve-se analisar o código e descobrir quais são os principais eventos e estados gerados por eles, ou quais são as variáveis importantes a serem monitoradas. Por exemplo, num programa de *ping pong* simples que apenas envia uma mensagem e espera por uma resposta, os principais eventos que ocorrem são o envio e a recepção dos dados. Assim, quando o programa inicia o evento de recepção de dados, a aplicação passa para o estado "esperando resposta", ou quando ele inicia um evento de envio de dados o programa passa para o estado de "enviando dados". Para que o rastreamento seja significativo, é importante também verificar a maneira com que é registrada a data da ocorrência dos eventos, ainda mais quando se trata de aplicações distribuídas onde os fluxos de execução estão em máquinas diferentes, com relógios diferentes. Para tratar este tipo de problema e facilitar o processo de geração de rastros é aconselhável a utilização de ferramentas especializadas, como a biblioteca de rastreamento libRastro [SIL 2002].

2.3 Visualização de Aplicações Concorrentes

O rastreamento de uma aplicação gera uma grande quantidade de dados necessários para descrever detalhadamente as mudanças de estados e os momentos onde há a maior demanda de recursos do sistema. Exibir estes dados de uma maneira clara para interpretação é uma tarefa difícil. Existem várias técnicas para analisar os dados monitorados, e a visualização é a maneira mais utilizada para mostrá-los de uma forma compreensível.

2.3.1 Ferramentas de Visualização de Aplicações Concorrentes

As ferramentas de visualização necessitam de três características básicas para proporcionar uma boa análise. A primeira delas é a interatividade, isto é, a ferramenta deve proporcionar ao usuário a capacidade de interagir com os dados para exibi-los da maneira que ele deseja. Além disso a ferramenta deve ser capaz de exibir mais detalhes de um objeto específico e de mostrar os vários tipos de dados em vários níveis de abstração diferentes. Isto é importante, pois é difícil interpretar todos os dados relevantes para uma depuração em uma única janela de dados. Outra característica importante é a extensibilidade. Ela define que a ferramenta deve permitir ao usuário a capacidade de agregar mais funcionalidades como processar diferentes tipos de dados, adicionar maneiras diferentes de visualização, possibilitar a interação com outras ferramentas, etc. A última característica é a escalabilidade (tanto de visualização quanto de processamento). É comum atualmente que aplicações sejam distribuídas em *clusters* de centenas de nós e que duram horas, ou mesmo dias, e nesses casos a quantidade de dados gerados é enorme. Uma boa ferramenta de visualização deve processar os dados e possibilitar uma visualização de forma legível e interativa, mesmo que a aplicação seja um sistema grande ou que tenha uma longa duração.

2.3.2 Ferramentas de Visualização Existentes

Várias ferramentas de visualização foram desenvolvidas para auxiliar a depuração de aplicações concorrentes. Alguns exemplos são as ferramentas ParaGraph, Pablo e Jumpshot-4.

2.3.2.1 ParaGraph

A ferramenta ParaGraph [HEA 2003] oferece ao usuário várias perspectivas de visualizações de aplicações executadas com MPI, com rastros no formato MPICL. Esta ferramenta não oferece uma boa interatividade, tem uma baixa escalabilidade e não possui extensibilidade devido ao fato de ter sido implementada de forma

monolítica.

2.3.2.2 Jumpshot-4

Jumpshot-4 [CHA 2004] (figura 2.1) é uma ferramenta de visualização de aplicações que utiliza arquivos de *log* no formato SLOG2 [PER 2004]. Este formato apresenta uma estrutura hierárquica de dados que possibilita uma visualização em vários níveis de abstração com uma boa escalabilidade de processamento. Além de possibilitar uma análise comportamental esta ferramenta possui a capacidade de gerar informações estatísticas da aplicação visualizada. A escalabilidade desta ferramenta é baseada no pré-processamento dos dados com uma indexação do arquivos de rastro.

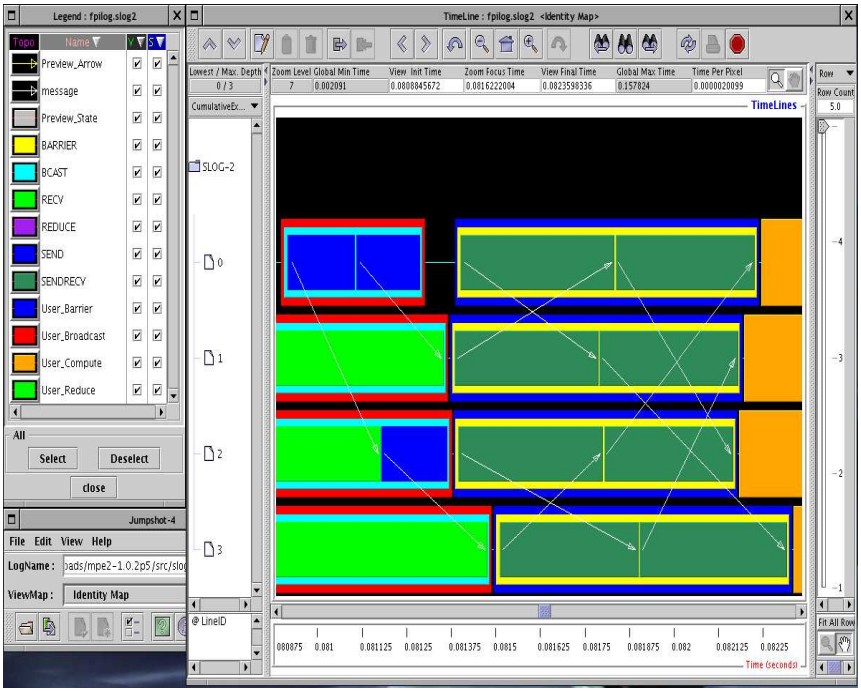

Figura 2.1: Interface gráfica da ferramenta Jumpshot-4.

2.3.2.3 Pablo

O ambiente de monitoração Pablo [REE 92] possui uma ferramenta gráfica para inserção de diretivas de monitoração na aplicação. Para ser extensível, Pablo foi desenvolvido como um grafo de fluxo de dados entre os componentes que podem

ser interligados, através de uma outra ferramenta gráfica, para produzir uma ferramenta de visualização específica. Esta ferramenta busca alcançar a escalabilidade através da troca de rastreamento para contagem de eventos quando a quantidade de dados monitorados se torna grande, provendo apenas visualizações sintéticas. Pablo não apresenta uma visualização comportamental, fornecendo apenas informações estatísticas.

Capítulo 3

Pajé - Ferramenta de Visualização de Rastros

Pajé é uma ferramenta de visualização de aplicações, em desenvolvimento no LSC/UFSM. É uma ferramenta genérica pois é independente de sistemas de monitoração e pode ser utilizada em uma grande variedade de contextos diferentes (visualização de um sistema *multi-threaded*, análise de fluxos de dados de um microprocessador, etc). A figura 3.1, por exemplo, exemplifica uma aplicação nos momentos onde um nó mestre (gppd) envia aos escravos dados para serem processados. Nas próximas subseções serão descritas as propriedades do Pajé de forma mais detalhada.

3.1 Extensibilidade

Para garantir sua extensibilidade, Pajé foi desenvolvido como de uma maneira modular (melhor analisada em seção posterior). Esta estrutura permite, por exemplo, alterar o formato do arquivo de leitura dos dados simplesmente alterando o módulo Leitor de Arquivos, sem precisar alterar os módulos subseqüentes. Esta estrutura modular permite que o presente trabalho possa ser desenvolvido alterando os módulos de gerenciamento de memória e simulação do Pajé sem ser necessário fazer alterações nos outros módulos. Outro fator que proporciona uma maior extensibilidade do Pajé é o fato dele ser uma ferramenta genérica, ou seja, que não depende de um sistema de monitoração ou modelo de programação específico. Assim, Pajé

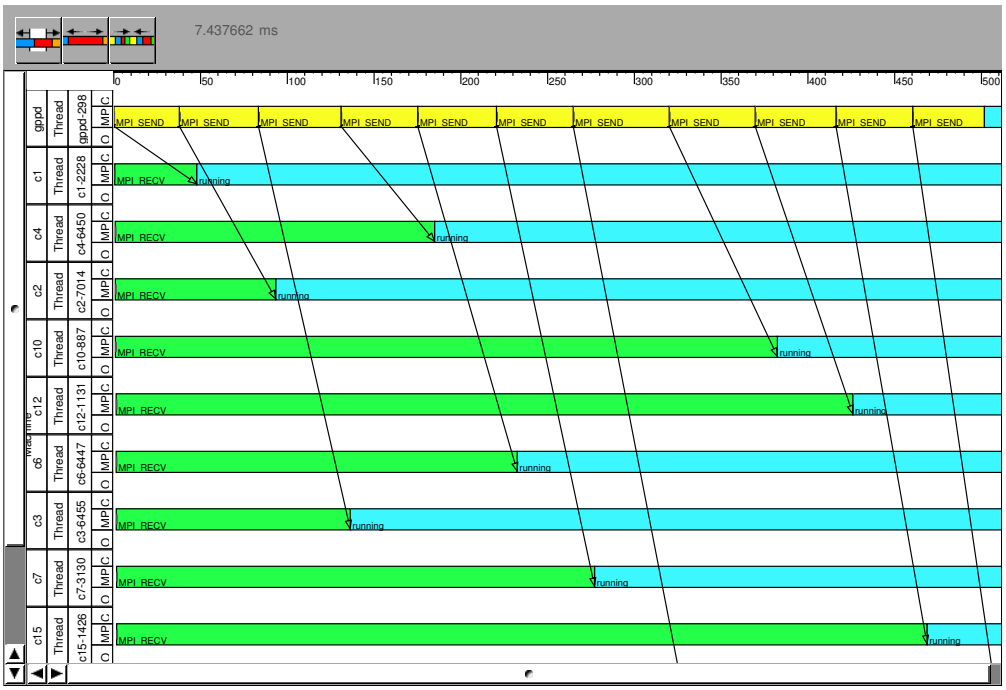

Figura 3.1: Janela de Espaço/Tempo do Pajé.

permite que o usuário defina o que deseja visualizar e a forma como os dados devem ser dispostos. O programador define isso através de uma árvore hierárquica de tipos de dados descrita no arquivo de entrada do Pajé.

3.2 Interatividade

O Pajé proporciona ao usuário a capacidade de alterar em diversas formas a maneira como os dados são exibidos na tela. Através de um conjunto de filtros, o desenvolvedor pode manipular as informações dos rastros, fazendo com que apenas o necessário seja visualizado. Por exemplo, o usuário pode analisar o rastro numa janela de espaço/tempo como mostra a figura 3.1 ou através de uma visualização estatística dos dados como mostra a figura 3.2. Além disso, o usuário pode selecionar uma entidade específica e, então, uma janela de informações será aberta, exibindo dados mais detalhados da entidade desejada. O usuário pode mover a janela de visualização temporal para frente e para trás, exibindo os dados graficamente de acordo com a ordem dos acontecimentos. Pajé possui vários filtros de alteração dos

dados exibidos que são interativos com o usuário.

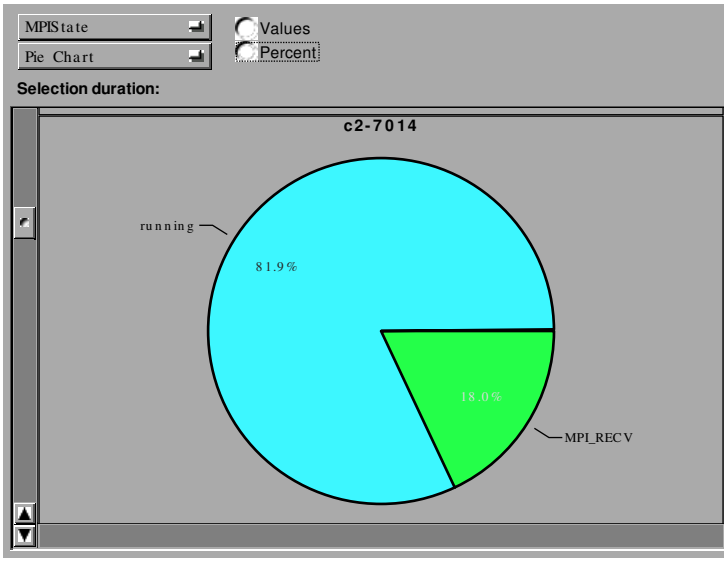

Figura 3.2: Visualização estatística de um rastro.

3.3 Escalabilidade

A fim de oferecer escalabilidade de visualização, Pajé possibilita a filtragem de dados que não sejam tão interessantes num certo momento da visualização, transformando uma grande quantidade de informações visuais em uma tela mais limpa. A partir daí o usuário pode escolher aumentar ou diminuir a quantidade de detalhes através do *zoom in/out* da visualização do rastro. Apesar disso, quando lidando com rastros grandes, estas características não são suficientes para prover uma visualização adequada.

3.4 Estrutura e Funcionamento

Pajé foi desenvolvido em um conjunto de módulos que se comunicam sob demanda. A figura 3.3 mostra a ligação entre componentes da ferramenta. O módulo **Controlador da Aplicação** responde pela interação com o usuário no menu principal da interface gráfica. O **Controlador de Rastro** é responsável por gerenciar os rastros abertos pela aplicação. O módulo **Leitor de Arquivos** é responsável por

retirar do arquivo de rastro as informações da aplicação visualizada. O **Simulador** transforma os dados brutos lidos pelo Leitor de Arquivos em objetos visualizáveis. O **Encapsulador** gerencia os objetos em memória. Os **Filtros** operam sobre os objetos visualizáveis, definindo o que deve ser visualizado. O **Visualizador** exibe na tela as informações requisitadas pelos usuários.

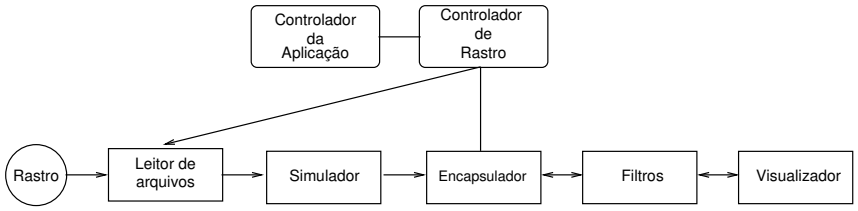

Figura 3.3: Estrutura modular do Pajé.

O funcionamento do Pajé decorre da seguinte maneira: O **Visualizador** faz uma requisição aos **Filtros** por dados que possam ser representados numa janela que começa em um tempo X e termina em um tempo Y. Os **Filtros** repassam esta requisição ao **Encapsulador**. Este, por sua vez, analisa os dados que ele possui em memória, e, caso ele tenha os dados requisitados, ele reponde a requisição. Caso contrário, ele avisa ao **Controlador de Rastro** que ele precisa de mais dados. O **Controlador de Rastro** envia uma mensagem ao **Leitor de Arquivos** que inicia a leitura de um *chunk*, que representa um pedaço do arquivo de rastro. Os dados lidos dos rastros são passados para o Simulador. O Simulador processa os dados e os passa para o Encapsulador que armazena os dados em uma estrutura hierárquica e responde a requisição. Quando os Filtros recebem uma resposta, inicia-se um processo de análise para retirar o que não precisa ser visualizado. Por fim os `Filtros` respondem à requisição do `Visualizador` que desenha na tela os dados recebidos. Este processo pode ser melhor analisado na figura 3.4. As etapas de simulação e armazenamento são bastante relevantes para este trabalho, assim como as requisições de dados. Estes processos estão melhor descritos nas subseções seguintes.

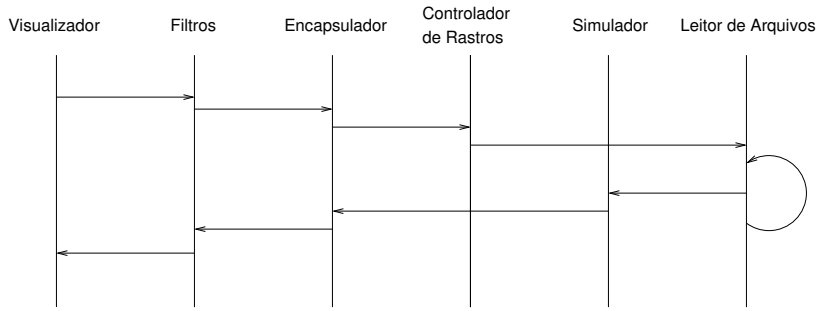

Figura 3.4: Fluxo de requisições do Pajé.

3.4.1 Classes de Dados Visualizáveis

Para melhor entendimento dos processos de simulação, armazenamento e requisição de dados, é preciso conhecer quais as classes de dados que representam a aplicação visualizada.

Existem basicamente seis classes de dados visualizáveis: os **eventos**, **estados**, **variáveis**, os ***links*** de comunicação, os **contêineres** e os **tipos**. Os **eventos** geralmente representam que algo significativo ocorreu na aplicação em um determinado instante, por exemplo, a aplicação recebeu um sinal de interrupção. Os **eventos** definem as alterações de **estados** da aplicação, o início e fim das comunicações e os valores das **variáveis**. Os **estados** representam a situação que a aplicação está durante um período específico, por exemplo, quando uma aplicação inicia uma chamada de recebimento de mensagem bloqueante ela entra no estado "bloqueado recebendo". Através dos **estados** pode-se analisar o comportamento da aplicação. Os *links* de comunicação representam o envio de mensagens entre os fluxos de execuções, através deles pode-se analisar, por exemplo, o tempo de duração das comunicações entre os processos. As **variáveis** representam os valores de atributos das aplicações, elas podem representar por exemplo o percentual de uso de *CPU* da aplicação durante um intervalo. Além destas classes existem também as entradas no arquivo de rastros que definem as entidades da aplicação, bem como os **tipos** e a hierarquia destas entidades. Estas entidades são representadas pelos **contêineres**. Cada **contêiner** possui um **tipo** que o define. Por exemplo, um contêiner pode ser o `fluxo 1` do tipo `thread`. Cada objeto visualizável possui uma data de inicio e fim e está con-

tido por um contêiner. A figura 3.5 demonstra o trecho de uma visualização. Nela pode-se identificar, por exemplo, que o **estado** `JVM State` pertence ao **contêiner** `JVM-369644` que é do **tipo** `Java Virtual Machine` e está contido no **contêiner** raiz `Programa Java`.

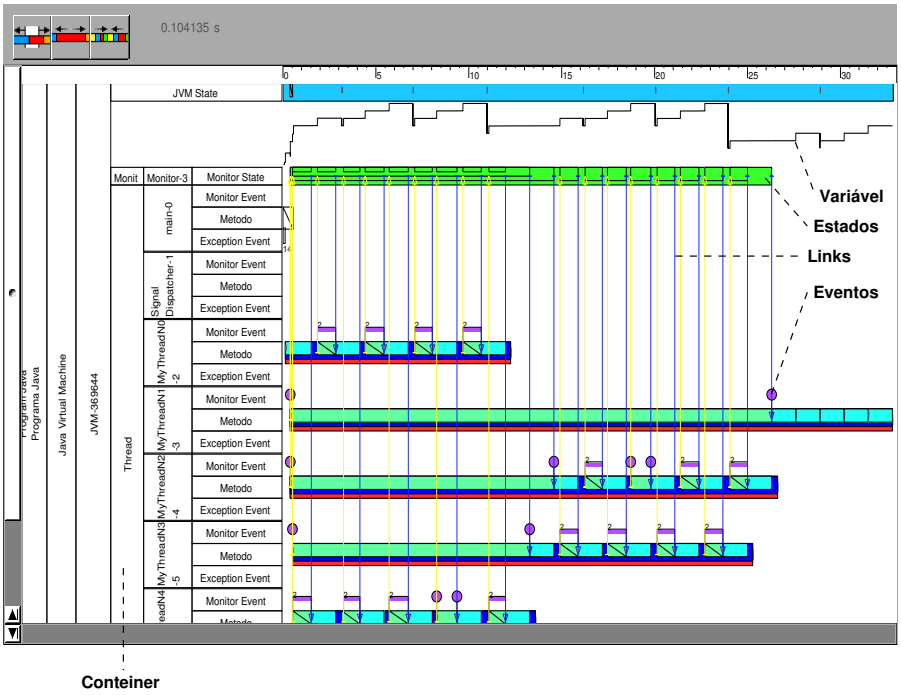

Figura 3.5: Exemplos das classes visualizáveis.

3.4.2 Simulação

O processo de simulação dos dados define os valores das classes de objetos visualizáveis. Isto ocorre da seguinte forma: primeiro o Simulador recebe os dados lidos do arquivo de rastro, em seguida ele procura em uma tabela de invocação qual o método a ser executado de acordo com tipo do dado lido do arquivo de rastro. Os métodos retornados obedecem a três padrões básicos - criação de entidades, definição de valores e destruição de entidades.

Os métodos de criação instanciam e inicializam novas entidades e as armazena, temporariamente durante o período de simulação, em dicionários. É bom ressaltar que os dicionários utilizados pelo Pajé são implementados como uma tabela *hash*, que utilizam como chave objetos passados por parâmetro no momento de inclusão

dos dados. Os métodos de definição buscam os objetos criados e os definem de acordo com valores recebidos dos rastro. Estes valores podem ser, por exemplo, os tempos de inicio e fim do objeto.

O método de destruição do Simulador finaliza um contêiner, definindo os valores finais dos objetos contidos pelo mesmo. Os objetos que possuem seus valores definidos são enviados ao Encapsulador para serem armazenados. O diagrama da figura 3.6 demonstra os passos da simulação mais freqüente que é quando o método retornado do mapa de invocação segue o padrão de definição de valores.

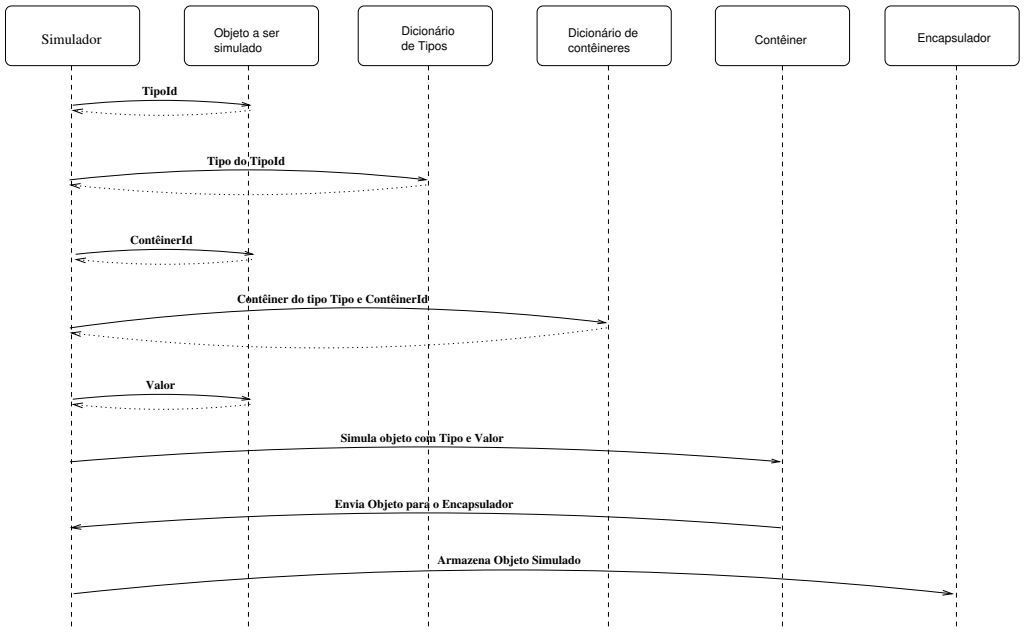

Figura 3.6: Diagrama de objetos de uma simulação.

3.4.3 Armazenamento

O armazenamento dos dados feito pelo Encapsulador segue uma hierarquia de três níveis que pode ser analisada na figura 3.7. O processo de armazenamento tem início quando um dado é enviado pelo Simulador. Os atributos do objeto a ser armazenado são analisados para o percorrimento da hierarquia. Os objetos são são depositados em listas temporais que estão indexadas em dicionários de acordo com o contêiner o qual elas pertencem. Os dicionários que contém as listas temporais estão armazenados em um segundo dicionário, cuja chave é o tipo do contêiner.

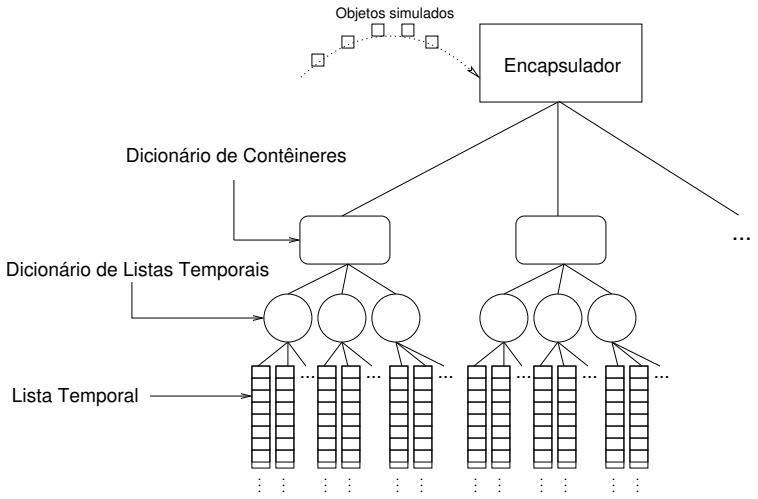

Figura 3.7: Hierarquia de armazenamento de dados.

Assim sendo, quando um objeto é recebido para ser armazenado, o Encapsulador analisa o tipo a quem o objeto pertence e procura no dicionário de contêineres pelo grupo de dados que possuem o mesmo atributo. Em seguida, uma pesquisa é feita no dicionário de listas temporais pela estrutura que contém os dados do mesmo contêiner do objeto a ser armazenado. O diagrama da figura 3.8 exemplifica o processo de armazenamento.

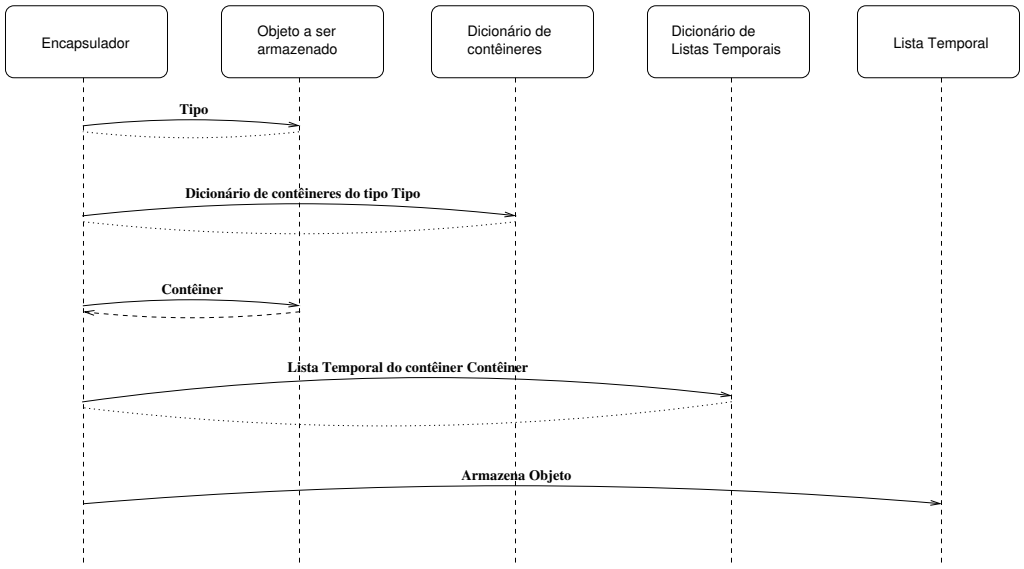

Figura 3.8: Diagrama de objetos de um armazenamento.

3.4.4 Requisições de Dados

As requisições por dados feitas pelo visualizador sempre especificam o tipo do objeto visualizável e o contêiner a quem ele pertence. Também são especificadas as datas início e fim do estado atual da janela de espaço/tempo. Desta forma, quando o Encapsulador recebe a requisição ele tem informação suficiente para percorrer a hierarquia de armazenamento até chegar na lista temporal dos objetos visualizáveis requisitados. Para tal, o Encapsulador procura em um dicionário de tipos pelo tipo especificado pelo parâmetro. Esta busca resulta em um dicionário de contêineres. Quando o dicionário de contêineres é identificado, o Encapsulador procura nele a lista temporal relativa ao contêiner requisitado pelo visualizador. A figura 3.9 demonstra o diagrama de objetos de uma requisição.

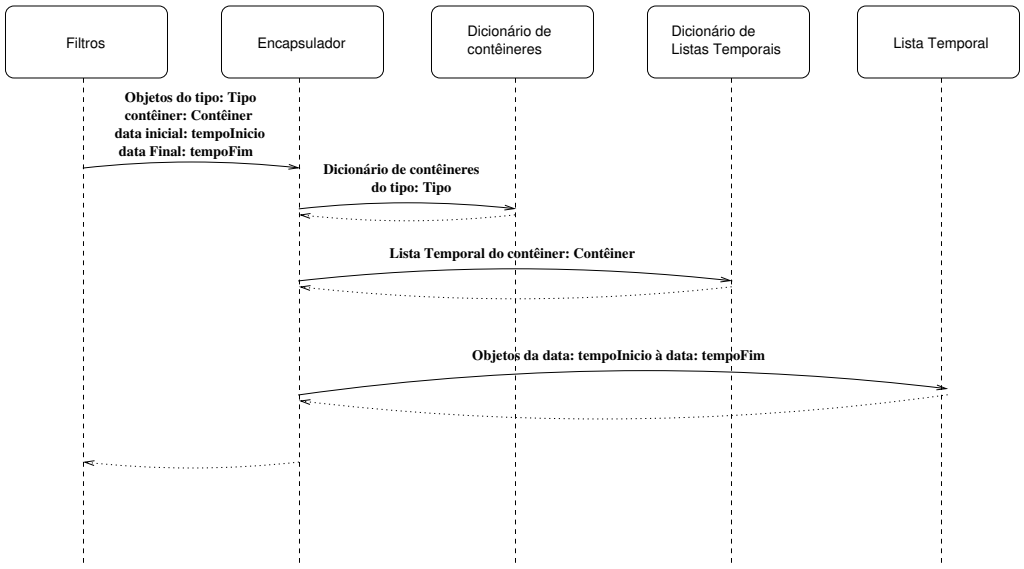

Figura 3.9: Diagrama de objetos de uma requisição.

Após a lista certa ser identificada, é iniciado um processo de busca para eliminar os objetos que não estão no intervalo da janela de espaço/tempo. Os objetos que devem ser retornados são àqueles que iniciam antes do tempo final e que terminam antes do tempo inicial da requisição. A lista temporal é composta por uma mapa de intervalos, onde os intervalos contêm um vetor de entidades ordenadas pelo tempo final. A figura 3.10 demonstra a estrutura da lista. Para encontrar objetos em uma

faixa de tempo nesta lista, primeiramente identifica-se o primeiro intervalo de tempo de acordo com a data de início. Através do intervalo de tempo encontra-se o vetor que contém os objetos visualizáveis. Estes objetos passam a ser selecionados a partir do tempo final, comparando o tempo final dos objetos com o tempo inicial requisitado pelo visualizador. Repete-se o processo até que o tempo inicial do intervalo seja maior que o tempo final requisitado.

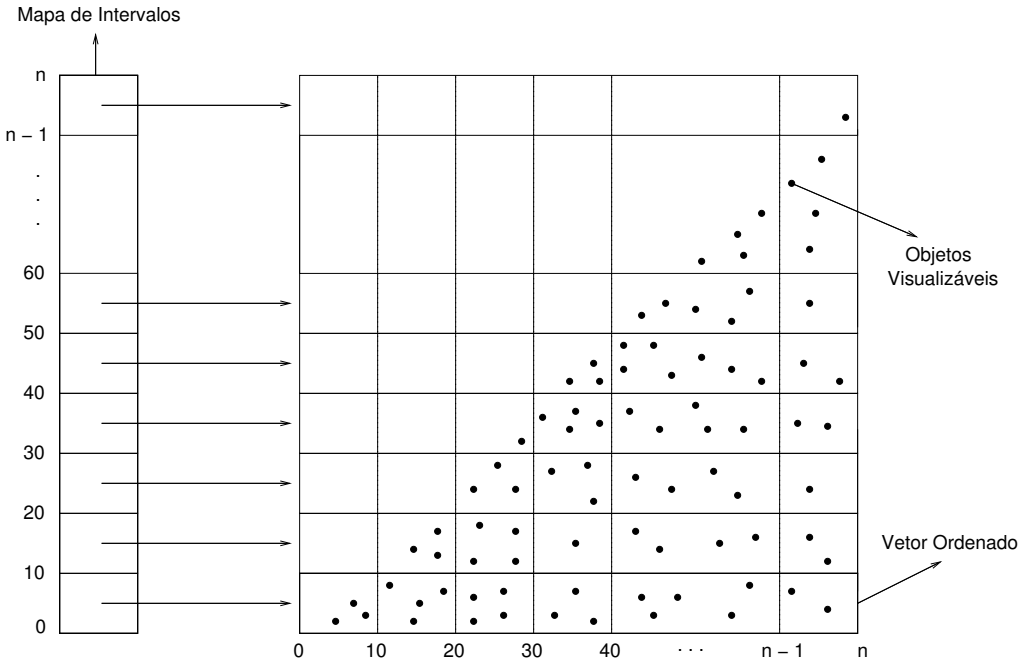

Figura 3.10: Estrutura da Lista Temporal

Após os dados desejados serem encontrados, eles são retornados aos `Filtros` que os repassam ao `Visualizador`. Para desenhar algumas das classes visualizáveis, o `Visualizador` precisa inverter a ordem dos objetos, para que os que terminem por último sejam desenhados primeiro. Dessa forma ele garante que quando alguns objetos sobrepõem outros, os que se encontram mais acima apareçam antes. Pode-se observar um exemplo de objetos sobrepostos na figura 3.5, onde alguns estados `Metodo` sobrepõem outros.

Capítulo 4

Reestruturação do Pajé

A reestruturação dos módulos de gerenciamento de memória e simulação do Pajé visou aumentar o desempenho da ferramenta na visualização de rastros.

Este capítulo inicia detalhando os problemas que motivaram este trabalho. Posteriormente, descreve os módulos alterados durante a implementação e, na seqüência, exibe as alterações realizadas.

4.1 Problemas de Escalabilidade

Este trabalho buscou resolver alguns problemas de desempenho que o Pajé possuía quando requisitada a visualização de rastros muito grandes. Estes problemas eram gerados, basicamente, por três fatores:

1. Sempre que o `Encapsulador` recebia um novo objeto ele tinha que percorrer uma hierarquia de três níveis para poder armazenar o dado recebido. Como o simulador envia um objeto por vez, o `Encapsulador` tinha que percorrer a hierarquia tantas vezes quanto o número de objetos visualizáveis criados;

2. A hierarquia de armazenamento fazia com que fosse preciso fazer buscas em dois dicionários para se chegar aos objetos de um certo tipo e contêiner. Isto aumentava a carga de processamento quando era necessária a exibição de muitos contêineres de vários tipos diferentes que possuem muitas classes de visualização diferentes;

3. Quando os eventos têm uma duração muito pequena e a janela de espaço/tempo está em uma escala muito grande, o `Encapsulador` retornava objetos que, muitas vezes, eram tão pequenos que ficavam difíceis de serem representados. A figura 4.1 mostra a visualização de uma aplicação numa escala muito grande em relação à duração dos estados. As linhas horizontais nessa figura são usadas pelo Pajé para representar seqüências de estados com duração curta demais para serem representadas na tela na escala desejada.

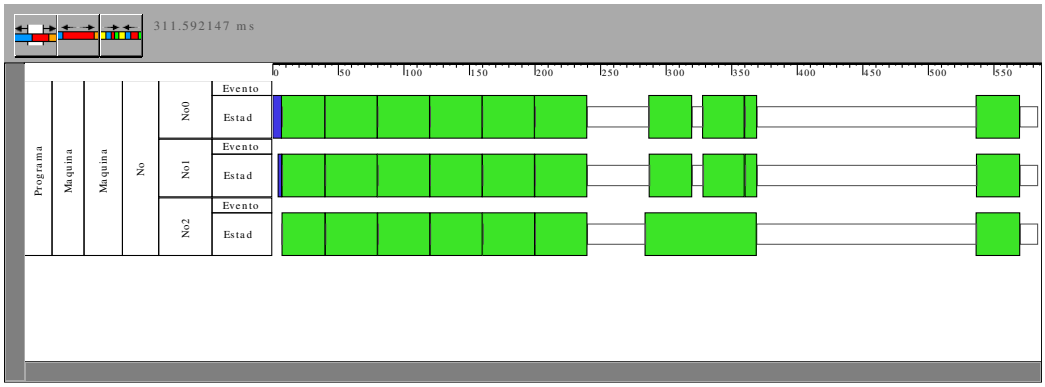

Figura 4.1: Problema de escalabilidade quando a escala de tempo é grande.

Nas subseções seguintes serão descritos mais detalhadamente os problemas de cada módulo.

4.1.1 Simulador

O `Simulador` possuía uma arquitetura de armazenamento de dados temporário parecida com a do `Encapsulador`, conseqüentemente ela possuía problemas similares. Sempre que uma nova entrada de rastros ia ser simulada, era preciso fazer buscas em dois dicionários para se chegar ao conjunto de dados do mesmo contêiner. Além disso, os objetos de classe de visualização diferentes do mesmo contêiner eram depositados em uma única estrutura, fazendo com que a busca por um objeto específico fosse mais demorada. Um outro problema encontrado neste módulo era o fato dele enviar os objetos simulados um a um ao `Encapsulador`, gerando uma carga maior para armazenamento dos dados.

4.1.2 Encapsulador

Os problemas encontrados no `Encapsulador` diziam respeito, principalmente, à hierarquia de armazenamento dos dados. Para cada novo objeto adicionado à memória era necessário a busca em dois dicionários para se chegar à Lista Temporal, mais uma pesquisa em um terceiro dicionário para encontrar o intervalo que o objeto se encaixava, e mais a adição do objeto em um vetor ordenado. Se analisar-se que um rastro comum tem cerca de 30.000 eventos que devem ser armazenados, chega-se ao número de 120.000 pesquisas em dicionários necessárias para armazenar hierarquicamente os dados. Além disso, alguns objetos armazenados não necessitam de uma estrutura complexa como a Lista Temporal para serem armazenados, pois eles não possuem duração, suas datas de início e fim são as mesmas. Nestes casos, o processamento pode ser simplificado. Outro problema do `Encapsulador` diz respeito aos objetos que eram retornados, mesmo eles tendo uma duração muito pequena para serem adequadamente representados, gerando uma carga de processamento muito grande para o `Visualizador` desenhar a janela de espaço/tempo.

4.2 Solução dos Problemas

Esta seção descreve as alterações realizadas no Pajé para solução dos problemas citados na seção anterior. Para melhor definição, as alterações são descritas de acordo com o módulo alterado.

4.2.1 Alterações no Simulador

Para aumentar o desempenho da simulação, parte da arquitetura de armazenamento temporário dos dados foi alterada. Para tal, foram criadas classes especializadas para simular cada classe de dados visualizáveis diferente, ou seja, foi criada uma classe que simula os estados, outra que simula os eventos, uma que simula as variáveis e, ainda, uma classe que simula os *links*. Os objetos dessas classes especializadas são armazenados em um dicionário e indexados com duas chaves: a identificação do contêiner e a identificação do tipo. Além disso, estas classes espe-

cializadas foram projetadas para armazenar os dados simulados até que o fim de um *chunk* fosse notificado, diminuindo a quantidade de vezes que o `Simulador` deve enviar informações ao `Encapsulador`.

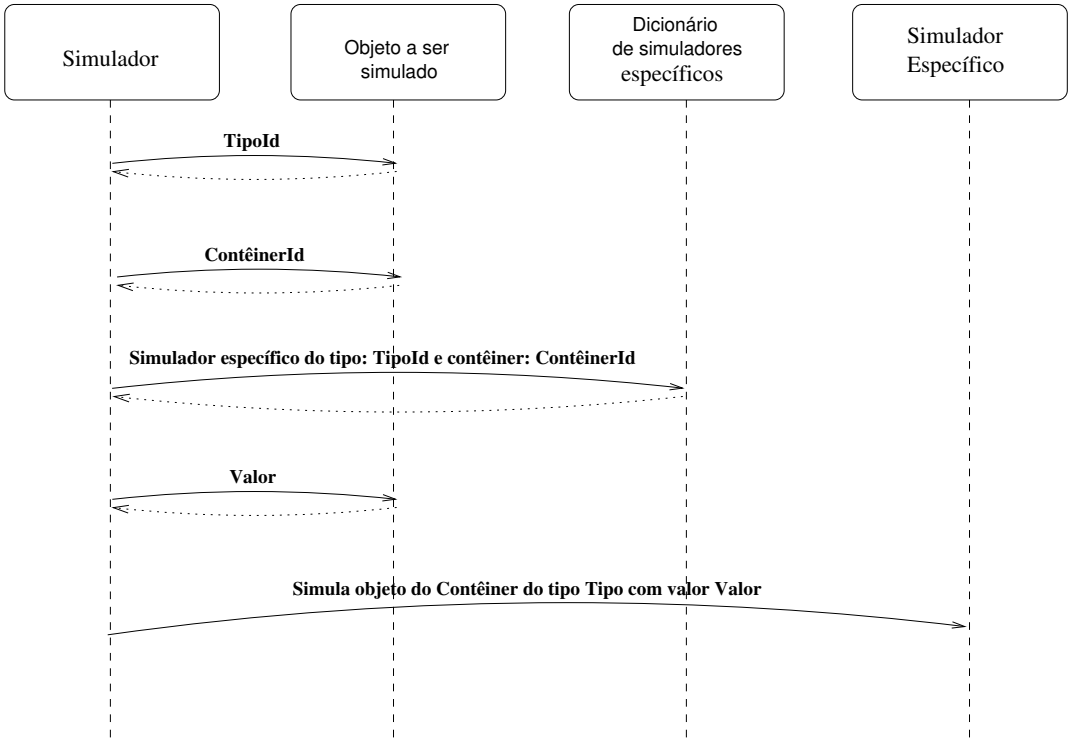

Figura 4.2: Novo processo de simulação

O processo de simulação foi alterado nos métodos que definem valores e naqueles que destroem contêineres. O diagrama da figura 4.2 demonstra a nova configuração da simulação dos dados exemplificada na figura 3.6. Agora, quando um objeto vai ser simulado, é feita uma busca em um dicionário de simuladores específicos utilizando a identificação do contêiner e do tipo como chaves. Caso a busca não retorne um objeto válido, o `Simulador` cria um novo simulador específico, de acordo com a classe visualizável do objeto que está sendo simulado, e o indexa no dicionário utilizando a mesma chave da busca. Com o simulador específico identificado, é iniciado a definição de valores. Esta etapa é feita utilizando métodos implementados para processarem objetos de apenas uma classe visualizável, eliminando rotinas e testes que eram necessários quando eram utilizados métodos mais genéricos. Uma pequena alteração foi feita no controlador de rastros para fazê-lo enviar uma mensagem ao

simulador, indicando o fim de um *chunk*.

Quando um fim de *chunk* é notificado, o simulador envia uma mensagem a todos os simuladores específicos. Estes, ao receber esta notificação, criam objetos de uma nova classe que representa os *chunks*. Os *chunks* criados recebem os objetos simulados e os armazenam. Existem vários tipos de *chunks* que correspondem as várias classes de dados visualizáveis. Por fim, os *chunks* criados são enviados para o `Encapsulador` para serem armazenados. A próxima subseção explica melhor as alterações no processo de armazenamento.

4.2.2 Alterações no Encapsulador

Os problemas encontrados neste módulo se relacionavam com a hierarquia de armazenamento dos dados e com a maneira que ele encontrava e retornava os objetos requisitados.

Para implementação das alterações neste módulo, foram utilizadas as classes que representam *chunks* de dados. Elas são capazes de identificar os objetos contidos nele, que pertencem a um certo intervalo de tempo. Apesar de existirem classes diferentes para cada classe visualizável, a interface de acesso aos dados obedece a um padrão, simplificando a maneira como ela é armazenada. Também foi criada uma classe que organiza *chunks* e é capaz de encontrar quais deles estão presentes em um determinado espaço de tempo.

O processo de armazenamento funciona agora da seguinte forma: quando um *chunk* é recebido, o `Encapsulador` analisa o tipo e o contêiner do *chunk*, e procura em um dicionário pelo organizador *chunks* que contém dados do mesmo tipo e contêiner. O *chunk*, então, é passado para o organizador encontrado, que o armazena em um vetor ordenado de acordo com o tempo final. A figura 4.3 exemplifica a nova hierarquia.

O `Encapsulador`, ao receber uma requisição, procura no dicionário de organizadores de *chunks* pelo responsável de armazenar os dados do tipo e contêiner recebidos pela requisição. Em seguida, uma requisição de objetos que pertencem ao intervalo de tempo pedido pelo `Visualizador` é enviada ao organizador de *chunks*

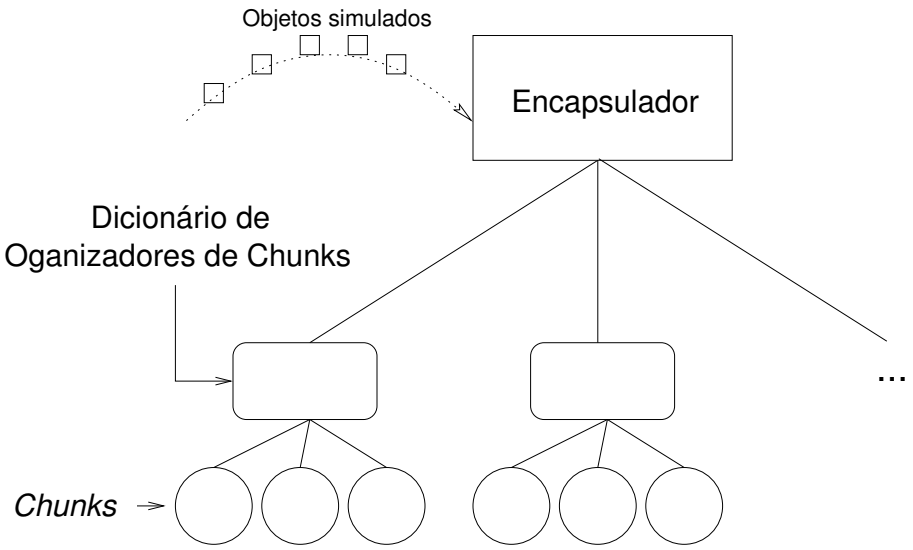

Figura 4.3: Nova hierarquia de armazenamento

encontrado. Este procura no vetor ordenado pelos *chunks* contidos no intervalo e faz uma requisição a cada deles pelos objetos que eles possuem. Para o último *chunk* do intervalo é pedido os objetos que iniciam antes do tempo final passado pela requisição. Para o primeiro, pede-se os objetos que se encontram depois do tempo inicial da requisição, e para aqueles que se encontram entre o primeiro e último *chunk*, são pedidos todos os objetos. Após os objetos serem retornados pelo organizador de *chunks*, o `Encapsulador` analisa o tempo de duração do objetos retornados e os compara com a duração mínima requisitada pelo `Visualizador`. Caso a duração do objeto seja menor, ele é incluído em um objeto visualizável virtual. Este objeto virtual passa a conter, então, objetos visualizáveis subseqüentes que não possuem uma duração maior que a requisitada pelo `Visualizador`. Quando a soma da duração dos objetos contidos em um objeto virtual supera a duração mínima requisitada, o objeto virtual é finalizado e outro é criado para continuar o processo, que continua até analisar todos os objetos retornados pelo organizador de *chunks*, realizando uma adaptação automática da visualização à escala de tempo. O `Visualizador` calcula a duração mínima de um objeto através de um parâmetro definido pelo usuário. É bom ressaltar que os objetos virtuais contêm um resumo dos objetos visualizáveis contidos nele.

Capítulo 5

Avaliação

Para fazer uma análise de impacto das alterações da nova implementação, foram escolhidos seis rastros com tamanhos variados. Os tipos das avaliações são descritos nas seções abaixo.

5.1 Tempo de Simulação

As alterações feitas no Pajé modificaram a maneira como os dados são simulados, então, uma avaliação foi feita para comparar em qual das implementações os rastros são simulados mais rapidamente. É interessante ressaltar que o tempo medido, em ambos os casos, inclui o período de leitura do arquivos de rastros. Para medir o tempo foram utilizados rastros que possuem uma quantidade de objetos visualizáveis que variam entre 238 e 1.776.209. O gráfico da figura 5.1 e a tabela 5.1 demonstram os resultados obtidos. A primeira linha da tabela corresponde ao tamanho dos arquivos de rastros, a segunda linha demonstra o tempo de simulação da antiga implementação do Pajé, a terceira linha corresponde ao tempo de simulação da nova implementação e a última linha mostra a relação entre a segunda e a terceira linha.

Analisando os resultados pode-se observar que a nova implementação do Pajé simula os gráficos mais lentamente. Isto acontece porque os dados são armazenados no `Simulador` até que a notificação de fim de *chunk* ocorra, gerando um processamento a mais que a implementação antiga não possuía.

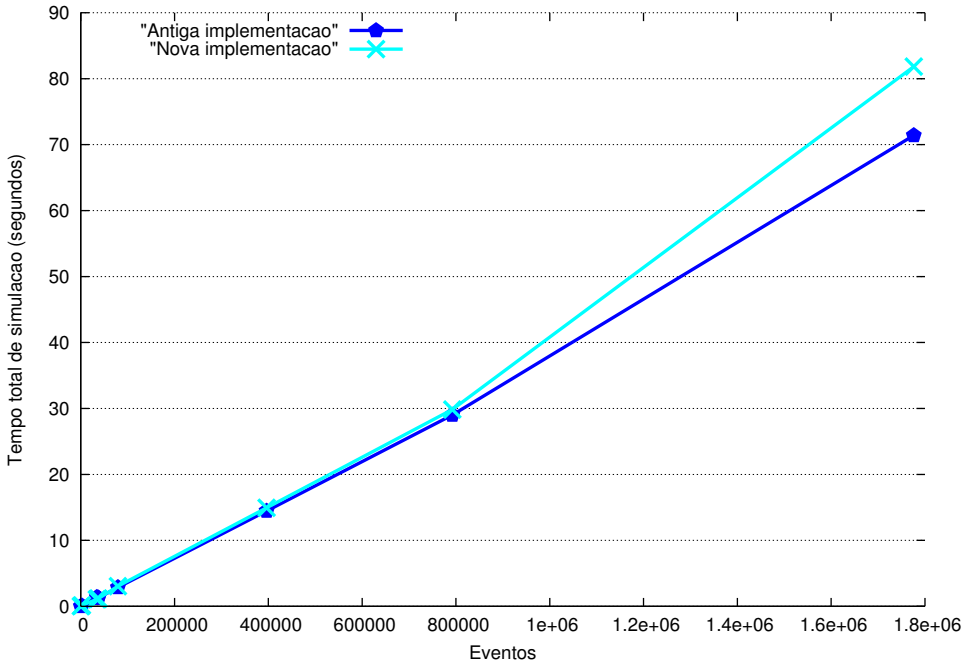

Figura 5.1: Comparação do tempo de simulação

Tabela 5.1: Comparação do tempo de simulação

Tamanho	2,5K	37K	79K	792K	1776K
T. A.	0,09s	1,2s	2,8s	29,0s	71,4s
T. N.	0,14s	1,1s	3,0s	29,8s	81,8s
	55,5%	-8,8%	5,9%	2,8%	14.6%

5.2 Tempo de Armazenamento

Para aumentar a escalabilidade do Pajé foi necessário alterar a hierarquia de armazenamento dos dados. Uma análise foi feita para comparar a diferença de tempo gasto para armazenar os objetos de um rastro entre a antiga e a nova implementação. Como o tempo de armazenamento é diretamente dependente da maneira como os dados são simulados e enviados ao Encapsulador, a medição do tempo foi feita incluindo o tempo de simulação de ambas as implementações. O gráfico da figura 5.2 e a tabela 5.2 representam os resultados da comparação.

Observando os resultados pode-se perceber que a nova implementação consegue armazenar os dados mais rapidamente, obtendo um ganho de cerca de 20%. Isto acontece em parte pela nova hierarquia de armazenamento e por ter diminuído a

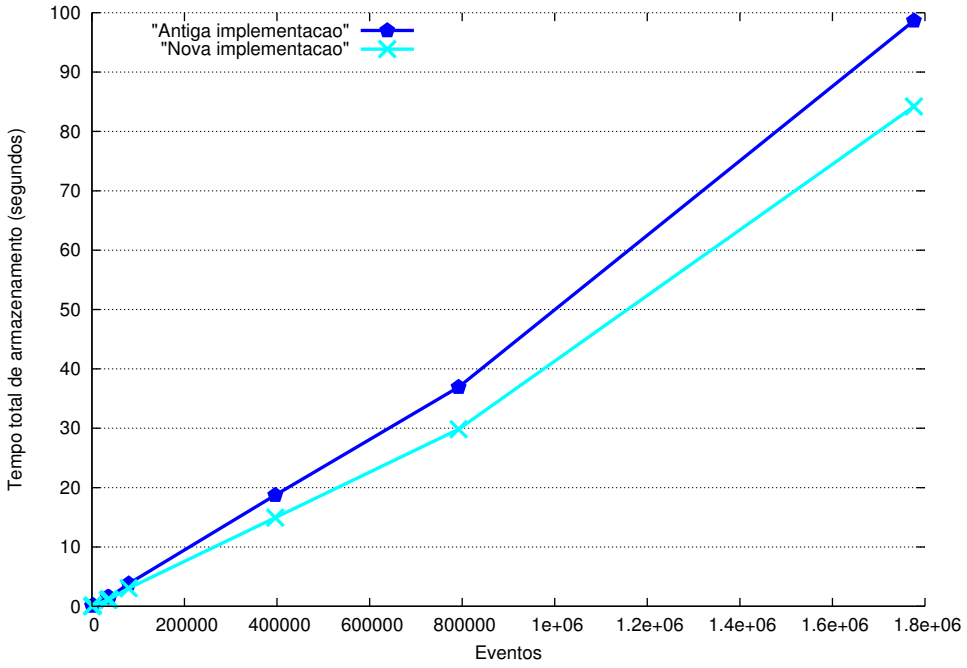

Figura 5.2: Comparação do tempo de armazenamento

Tabela 5.2: Comparação do tempo de armazenamento

Tamanho	2,5K	37K	79K	792K	1776K
T. A.	0,19s	1,5s	3,8s	36,9s	98,6s
T. N.	0,14s	1,1s	3,1s	29,8s	84,2s
	-26,3%	-22,3%	-19,7%	-19,2%	-14.6%

quantidade de vezes que o `Encapsulador` precisa executar o processo de armazenamento, pois os dados agora são enviados em blocos. Pode-se concluir, também, que o aumento de desempenho nesta etapa compensou o fato da simulação estar mais lenta.

5.3 Tempo de Resposta a uma Requisição

A maneira como os dados são consultados na memória foi modificada devido às alterações na hierarquia de armazenamento dos objetos visualizáveis. Por isso, uma avaliação do tempo necessário para enumerar os objetos requisitados pelo `Visualizador`. A medição do tempo foi feita no momento que é iniciada a requisição de todos os objetos do rastro para gerar uma visualização geral da aplicação.

Tabela 5.3: Comparação do tempo de resposta à requisições

Tamanho	13	133	4,4K	22K	44K
T. A.	1,3μs	1,8μs	2,0μs	2,3μs	2,6μs
T. N.	1,2μs	1,6μs	1,6μs	1,6μs	2,0μs
	-7,1%	-10,5%	-19,0%	-27,6%	-22.6%

Os resultados obtidos estão dispostos na figura 5.3 e na tabela 5.3.

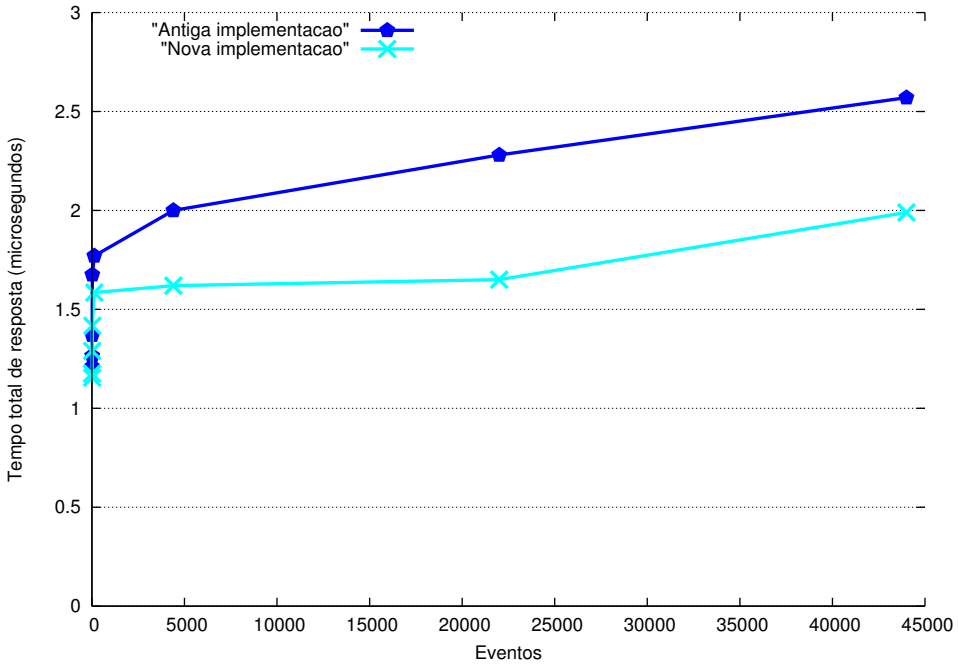

Figura 5.3: Comparação do tempo de resposta de uma requisição

Analisando a comparação, percebe-se que a nova implementação do Pajé conseguiu diminuir o tempo de resposta a uma requisição. Isto aconteceu porque os objetos estão agora armazenado em estruturas específicas para seu tipo de dado. Além disso, a nova hierarquia da memória possibilita um acesso mais direto aos dados desejados.

5.4 Adaptação Automática à Escala de Tempo

Para aumentar a escalabilidade de visualização da ferramenta Pajé foi desenvolvido um sistema de adaptação dos objetos visualizáveis à escala de tempo. Este

Figura 5.4: Estados virtuais que representam os resumos dos estados que não são visíveis em grandes escalas.

novo sistema gera visualizações menos poluídas e mais compreensíveis. A figura 5.4 demonstra a mesma visualização da figura 4.1 utilizando o sistema de adaptação automática de dados. Nessa visualização, uma seqüência de estados de depuração muito curta é representada como um estado que agrupa os estados curtos demais de acordo com o seu valor. Para distinguir esses estados agrupados dos estados normais, eles são representados com uma linha diagonal.

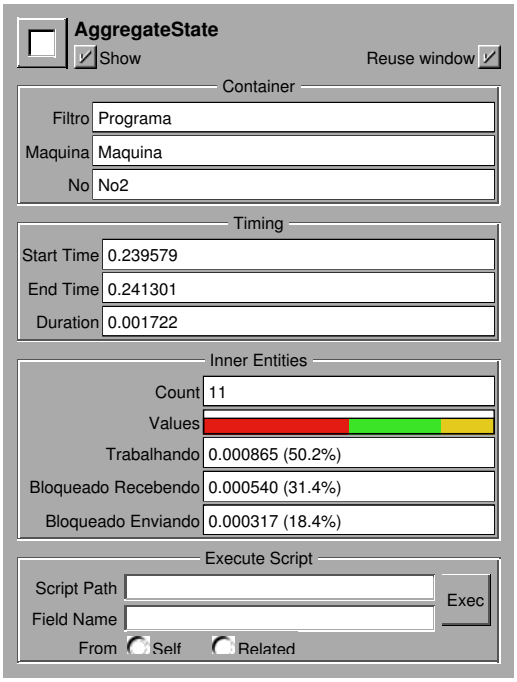

Figura 5.5: Janela de informações de um estado agrupado

Para auxiliar a análise dos estados agrupados, uma janela de informações (figura 5.5) pode ser aberta pelo usuário. Esta janela contém o resumo dos estados agrupados, exibindo a soma das durações de cada estado, e a relação dessa soma com o tempo total do estado agrupado.

Capítulo 6

Conclusão

A visualização de rastros de aplicações pode auxiliar na depuração de aplicações concorrentes e existem várias ferramentas para este tipo de visualização. Pajé é uma ferramenta que possibilita uma análise comportamental, exibindo em uma tela os estados da aplicação assumidos durante a execução.

Este trabalho apresentou algumas alterações realizadas na ferramenta Pajé para aumentar a escalabilidade de processamento e visualização durante a exibição de rastros de aplicações. A análise destas alterações mostrou que a nova implementação possibilitou uma melhoria no desempenho da ferramenta durante o processamento dos rastro. Concluiu-se também que o sistema de adaptação automática dos dados incluído no Pajé possibilita uma visualização mais clara de aplicações, mesmo quando a escala de tempo utilizada é muito grande em relação a duração de seus estados.

Trabalhos Futuros

Como trabalhos futuros pretende-se agregar às alterações realizadas na ferramenta Pajé ao sistema de *checkpoint* da ferramenta. Este sistema garante a ferramenta Pajé um melhor gerenciamento dos dados, mantendo na memória apenas parte dos objetos interpretados do arquivo de rastro.

Existe também a intenção de criar um módulo adicional ao Pajé que funciona como uma *cache* de dados, responsável por manter disponível ao Visualizador

objetos visualizáveis já processados, diminuindo a quantidade de requisições feitas ao Encapsulador, e, por conseguinte, diminuindo a sobrecarga de processamento da ferramenta.

Referências Bibliográficas

[CHA 2004] CHAN, A. et al. **Jumpshot-4 users guide**. [S.l.: s.n.], 2004. `http://www-unix.mcs.anl.gov/perfvis/software/viewers/jumpshot-4/`.

[HEA 2003] HEATH, M. T.; FINGER, J. E. **Paragraph**: a performance visualization tool for mpi. [S.l.: s.n.], 2003. `http://inf-server.inf.uth.gr/courses/CE650/material/userguidePG.pdf`.

[KER 2000] KERGOMMEAUX, J. de; STEIN, B. **Pajé, an extensible and interactive and scalable environment for visualizing parallel programs executions**. [S.l.]: Unité de recherche INRIA Rhône-Alpes, 2000. `http://www.inria.fr`.

[OLI 2000] OLIVEIRA STEIN, B. de; KERGOMMEAUX, J. C. de; BERNARD, P. É. Pajé, an interactive and visual tool for tuning multi-threaded parallel applications. **Parallel Computing**, v.26, n.10, p.1253–1274, Aug. 2000.

[OLI 99] OLIVEIRA STEIN, B. de. **Visualisation interactive et extensible de programmes parallèles à base de processus légers**. 1999. Thèse de doctorat en informatique — Université Joseph Fourier, France.

[PER 2004] PERFORMANCE visualization for parallel programs. [S.l.: s.n.], 2004. `http://www-unix.mcs.anl.gov/perfvis/software/viewers/jumpshot-4/`.

[REE 92] REED, D. A. et al. **An overview of the pablo performance analysis environment**. [S.l.]: University of Illinois, 1992.

[SIL 2002] SILVA, G. J. da; STEIN, B. Uma Biblioteca Genérica de Geração de Rastros de Execução para Visualização de Programas. **Anais do I Simpósio de Informática da Região Centro, Santa Maria**, 2002.