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

RAFAEL KELLER TESSER

Monitoramento On-line em Sistemas Distribuídos: Mecanismo Hierárquico Para Coleta de Dados

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

Prof. Dr. Philippe Olivier Alexandre Navaux
Orientador

Porto Alegre, Agosto de 2011

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Tesser, Rafael Keller

Monitoramento On-line em Sistemas Distribuídos: Mecanismo Hierárquico Para Coleta de Dados / Rafael Keller Tesser. – Porto Alegre: PPGC da UFRGS, 2011.

106 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2011. Orientador: Philippe Olivier Alexandre Navaux.

1. Monitoramento. 2. Sistemas distribuídos. 3. Programas distribuídos. 4. Coleta de dados. 5. Análise de comportamento. 6. Visualização de informações. I. Navaux, Philippe Olivier Alexandre. II. Título.

“One accurate measurement is worth a thousand expert opinions.”
— ADMIRAL GRACE HOOPER

AGRADECIMENTOS

Agradeço ao professor Philippe Olivier Alexandre Navaux, meu orientador. A Lucas Mello Schnorr, desenvolvedor do DIMVisual e do TRIVA, pelo auxílio no seu estudo e na realização modificações necessárias a eles. Agradeço também aos colegas da sala 209 do prédio 67 do Instituto de Informática, pela amizade e por proporcionar um ambiente de trabalho agradável.

Agradeço a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo auxílio financeiro através da concessão de bolsas, durante a realização desse trabalho.

Experimentos apresentados nessa dissertação foram realizados utilizando recursos do Grid'5000, que é desenvolvido como parte do projeto INRIA ALADDIN, com suporte do CNRS, RENATER e várias universidades, assim como outras organizações de fomento (ver https://www.grid5000.fr).

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 9

LISTA DE FIGURAS 11

LISTA DE TABELAS 13

RESUMO 15

ABSTRACT 17

1 INTRODUÇÃO 19
1.1 Monitoramento de Sistemas Distribuídos 20
1.2 Apresentação do Trabalho 22
1.3 Organização do texto 24
1.4 Considerações Finais 24

2 MONITORAMENTO DE SISTEMAS DISTRIBUíDOS 25
2.1 Etapas do Monitoramento de Sistemas Distribuídos 25
2.1.1 Geração dos Dados de Monitoramento 25
2.1.2 Processamento de Dados de Monitoramento 28
2.1.3 Distribuição de Informações de Monitoramento 31
2.1.4 Apresentação de Informações de Monitoramento 31
2.2 Aspectos Relacionados à Implementação 32
2.2.1 Intrusividade do Sistema de Monitoramento 33
2.2.2 Estado Global, Tempo e Ordenação de Eventos 34
2.3 Representação dos Dados de Monitoramento 35
2.3.1 Contadores e Temporizadores 35
2.3.2 Rastros 36
2.3.3 Vetores: Séries temporais 36
2.4 Monitoramento de Grid 36
2.4.1 Requisitos para ferramentas de monitoramento de *Grid* 37
2.4.2 The Grid Monitoring Architecture 38
2.4.3 A taxonomia de Zanikolas e Sakellariou 39
2.5 Considerações Finais 40

3 FERRAMENTAS DE MONITORAMENTO 43
3.1 DIMVisual 43
3.2 TRIVA 44
3.3 Estudo de Sistemas de Monitoramento 45

3.3.1 Características Analisadas 46
3.3.2 Ganglia 48
3.3.3 GridICE 49
3.3.4 MonALISA 51
3.3.5 Globus Monitoring And Discovery Service 52
3.3.6 Monalytics 55
3.4 Considerações Finais 56

4 DIMVCHM: PROPOSTA DE UM MODELO HIERÁRQUICO DE COLETA DE DADOS DE MONITORAMENTO 57
4.1 Componentes 59
4.1.1 Coletores 59
4.1.2 Agregadores 60
4.1.3 Clientes 60
4.2 Registro de Coletores 60
4.3 Subscrição 61
4.4 Transferência de dados 62
4.5 Monitoramento utilizando o DIMVHCM 62
4.6 Comparação de Ferramentas de Monitoramento 63
4.7 Outras possibilidades 66
4.8 Considerações Finais 67

5 IMPLEMENTAÇÃO DO PROTÓTIPO 69
5.1 DIMVisual Hierarchichal Collection Model 70
5.1.1 Subscrição 75
5.2 Implementação do Cliente 78
5.2.1 Integração ao DIMVisual 78
5.2.2 Integração com o TRIVA 80
5.3 Considerações Finais 84

6 TESTES E AVALIAÇÃO 87
6.1 Duração da transmissão de dados de monitoramento 87
6.2 Avaliação do processamento dos dados no cliente 95
6.3 Considerações finais 98

7 CONCLUSÕES E TRABALHOS FUTUROS 101

REFERÊNCIAS 103

LISTA DE ABREVIATURAS E SIGLAS

API	Aplication Programming Interface
DDSA	Dynamic Distributed Services Architecture
DIMVHCM	DIMVisual Hierarchical Collection Model
DIMVisual	Data Integration Model for Visualization
GIIS	Grid Index Information Service
GIS	Grid Information Service
GMA	Grid Monitoring Architecture
GOC	Grid Operation Center
GRIP	Grid Information Protocol
GRIS	Grid Resource Information Service
GRRP	Grid Registration Protocol
GT4	Globus Toolkit versão 4
JINI	Java Intelligent Network Infrastructure
LDAP	Lightweight Directory Acces Protocol
LUS	Lookup Service
MDS	Monitoring and Discovery Service
MonALISA	Monitoring Agents in a Large Scale Integrated Service Architecture
NWS	Network Weather Service
OGF	Open Grid Forum
RRDtool	Round Robin Database tool
SOAP	Simple Object Acces Protocol
TRIVA	Three Dimensional Interactive Visualization Analysis
VO	Virtual Organization
WSDL	Web Services Description Language
WSRF	Web Services Resource Framework
XML	Extended Markup Language
XSLT	Extensible Stylesheet Language Transformations

LISTA DE FIGURAS

Figura 1.1: Diagrama dos tipos de tarefas realizadas por um sistema de monitoramento. 21

Figura 2.1: Diagrama das tarefas relacionadas à geração de dados dados de monitoramento 26
Figura 2.2: Diagrama das tarefas relacionadas ao processamento de dados de monitoramento 29
Figura 2.3: Relacionamento entre a visão do sistema de monitoramento por componentes e as funcionalidades citadas no capítulo 1. 37
Figura 2.4: Componentes da GMA (Grid Monitoring Architecture) 39
Figura 2.5: Níveis da Taxonomia de Zanikolas e Sakellariou. 40

Figura 3.1: Modelo da integração de dados no DIMVisual 43
Figura 3.2: Arquitetura do protótipo do DIMVisual 44
Figura 3.3: Diagrama dos componentes do TRIVA 46
Figura 3.4: Arquitetura da ferramenta de monitoramento Ganglia, mostrando os serviços *gmond* e *gmetad* 48
Figura 3.5: Camadas da arquitetura da ferramenta de monitoramento GridICE . . 50
Figura 3.6: Interface gráfica *Web Start* da ferramenta MonALISA 52
Figura 3.7: Exemplo de estrutura hierárquica formada por *Aggregate Directories* 54

Figura 4.1: Grid (multicluster) visto como conjunto e como hierarquia. 58
Figura 4.2: Componentes do Modelo Hierárquico de Coleta de Dados de Monitoramento 59
Figura 4.3: Exemplo de organização dos componentes do DIMVHCM em um sistema de monitoramento. 63

Figura 5.1: Componentes do Protótipo Implementado 69
Figura 5.2: Hierarquia do modelo de coleta mostrando as classes implementadas . 71
Figura 5.3: Fluxograma da inicialização de um `DIMVCollector` 71
Figura 5.4: Fluxograma da thread de conexão de um `DIMVCollector` 72
Figura 5.5: Fluxograma do envio de dados por um `DIMVCollector` 73
Figura 5.6: Fluxograma da inicialização de um `DIMVAggregator` 74
Figura 5.7: Fluxograma do recebimento de dados por um `DIMVAggregator` . 75
Figura 5.8: Fluxograma da inicialização de um `DIMVClient` 75
Figura 5.9: Fluxograma da inicialização da thread de conexão de um `DIMVClient` 76
Figura 5.10: Fluxograma do recebimento de dados por um `DIMVClient` 76

Figura 5.11: Fluxograma do processamento de uma solicitação de subscrição por um DIMVAggregator 77
Figura 5.12: Fluxograma do processamento de uma solicitação de subscrição por um DIMVClient 78
Figura 5.13: Fluxograma do envio de uma solicitação de subscrição na thread de conexão de um DIMVClient 78
Figura 5.14: Fluxograma da solicitação de subscrição por um DIMVCollector . 78
Figura 5.15: Arquitetura do Cliente Implementado 79
Figura 5.16: Fluxograma do recebimento de dados por uma fonte de dados do DIMVisual 79
Figura 5.17: Fluxograma do fornecimento de dados convertidos pelo componente Order do DIMVisual 80
Figura 5.18: Diagrama de componentes do TRIVA com o DIMVHCMReader . . . 81
Figura 5.19: Fluxograma da inicialização do TRIVADIMVHCMController . . 82
Figura 5.20: Fluxograma da thread produtora do DIMVHCMReader 83
Figura 5.21: Fluxograma da thread consumidora no TRIVADIMVHCMController 84
Figura 5.22: Fluxograma da leitura de um *chunk* pelo DIMVHCMReader 85

Figura 6.1: Hierarquias utilizadas nos experimentos 88
Figura 6.2: Duração dos envios utilizando 80 coletores 90
Figura 6.3: Duração média dos envios com 160 coletores 90
Figura 6.4: Duração média dos envios com 320 coletores 90
Figura 6.5: Duração média dos envios para as quatro configurações dos coletores. 92
Figura 6.6: Tempo para envio de dados entre os níveis da hierarquia 1ag1cli . . . 94
Figura 6.7: Tempo para envio de dados entre os níveis da hierarquia 4ag1cli . . . 95
Figura 6.8: Tempo para envio de dados entre os níveis da hierarquia 4plus1ag1cli 95
Figura 6.9: DIMVisual: Frequências de Recebimento x Consumo de dados 97
Figura 6.10: DIMVisual: Frequências de Leitura x Conversão de Eventos 98
Figura 6.11: DIMVHCM Reader: Frequências de Recebimento x Consumo de Eventos 99
Figura 6.12: TRIVA: Frequências de Leitura de *Chunks* x Atualização da Visualização 100

LISTA DE TABELAS

Tabela 4.1: Comparação das ferramentas em relação à distribuição das informações de monitoramento 64
Tabela 4.2: Comparação das ferramentas em relação ao armazenamento de dados históricos 65
Tabela 4.3: Comparação das ferramentas em relação à apresentação das informações de monitoramento 66
Tabela 4.4: Comparação das ferramentas segundo classificação na taxonomia de Zanikolas e Sakellariou 67

Tabela 6.1: Duração média dos envios com erros referentes ao intervalo de confiança de 95%. 91
Tabela 6.2: Ganho pela agregação de dados nas mensagens (5 coletas/mensagem). 91
Tabela 6.3: Diferenças de tempo com as diferentes quantias de coletores. 93
Tabela 6.4: Diferenças de tempo entre as diferentes hierarquias. 94

RESUMO

Este trabalho propõe um modelo hierárquico para coleta de dados de monitoramento em sistemas distribuídos. Seu objetivo é proporcionar a análise *on-line* do comportamento de sistemas e programas distribuídos. O meio escolhido para realizar essa análise foi a visualização. Inicialmente é apresentada uma contextualização sobre monitoramento de sistemas distribuídos. Também são abordados aspectos específicos ao monitoramento de *Grid*. Após, é analisado um conjunto de ferramentas de monitoramento. Então tem-se a apresentação do modelo proposto. Esse é composto por coletores locais, por uma hierarquia de agregadores e por clientes. É utilizado o modelo *push* de transmissão de dados e há um mecanismo de subscrição aos coletores. Foi implementado um protótipo do modelo de coleta proposto, que foi utilizado na implementação de um protótipo de ferramenta de monitoramento *on-line*. Nessa, os dados coletados são fornecidos ao DIM-Visual, que é um modelo de integração de dados para visualização. Para visualização, o protótipo utiliza a ferramenta TRIVA, que recebe os dados integrados como entrada. Essa ferramenta foi modificada para gerar uma visualização que é atualizada de maneira *on-line*. Também foram realizados experimentos para avaliar o tempo necessário para enviar mensagens com diferentes hierarquias e configurações dos coletores. Além disso, foi avaliada a capacidade de o cliente implementado processar os dados recebidos, gerando sua visualização.

Palavras-chave: Monitoramento, sistemas distribuídos, programas distribuídos, coleta de dados, análise de comportamento, visualização de informações.

On-line Monitoring of Distributed Systems: A Hierarchical Mechanism for Data Collection

ABSTRACT

This work proposes a hierarchical model for collecting monitoring data from distributed systems. Its goal is to allow the on-line analysis of the behavior of distributed systems and applications. The means we chose to perform this analysis is to generate a visualization of the collected information. In the beginning of this dissertation we present an overview of the monitoring of distributed systems. Aspects that are specific to the monitoring of *Grid* systems are also reviewed. Next, we have an analysis of a set of monitoring tools. Then we present the proposed model, which is composed by local collectors, an hierarchical structure of aggregators and clients. A push data transmission model is used in the model and it also has a subscription mechanism. A prototype monitoring tool was implemented, integrating the data collection model with DIMVisual and TRIVA. The former is a data integration model whose output is formatted to be used as input for a visualization tool. The later is a visualization tool which, in the prototype, receives the integrated data from DIMVisual. TRIVA generates a visualization of the received information, which is updated in an on-line fashion. In order to evaluate the model, we performed a set of experiments using the prototype. One of the experiments measured the time spent to send data though different hierarchies. In these tests we have also varied the quantity and the configuration of the collectors. In another experiment we evaluated the capacity of the client to process the received data.

Keywords: monitoring, distributed systems, distributed applications, data collection, behavioral analysis, information visualization.

1 INTRODUÇÃO

Sistemas de computadores podem tirar grande proveito de sistemas de monitoramento. Eles podem fornecer várias informações, como o estado dos componentes do sistema, dados sobre a execução de aplicações ou sobre o funcionamento da rede de interconexão. Os dados de monitoramento podem ser usados para múltiplas tarefas, incluindo: detecção de falhas, escalonamento, depuração e análise de desempenho de aplicações. É possível também armazenar dados históricos, a fim de analisar o comportamento do sistema ao longo de um determinado período de tempo.

Um sistema de monitoramento pode ser utilizado para obtenção de variáveis que representem o estado do sistema em determinado instante. Um exemplo de uso dessas informações seria a tomada de decisões de escalonamento, tendo como base a carga atual de cada recurso disponível. No entanto, para algumas tarefas, como a depuração de programas, o desejado é uma forma de analisar o comportamento do sistema ou do programa durante um determinado intervalo de tempo. Ou seja, quer-se observar as mudanças do estado do sistema e do programa, ocorridas durante a execução do mesmo. Para isso é interessante ter a possibilidade de obter uma sequência de dados, gerada ao longo do intervalo que se quer observar.

No segundo caso citado, pode-se classificar o processo de monitoramento de acordo com quando os dados obtidos tornam-se disponíveis. Nesse caso, os dados podem ser apresentados após o término do processo observado (*post-mortem*) ou em tempo de execução (*on-line*). No caso *post-mortem*, o processamento e a análise dos dados não concorrem por recursos com o sistema ou programa sendo observado. Por isso, esse esquema tem a vantagem de reduzir a intrusividade do monitoramento. No entanto, em alguns casos, pode ser interessante observar mudanças no comportamento de maneira mais imediata. Por exemplo, poder-se-ia estar acompanhando a execução de um programa que tenha uma longa duração. Prováveis objetivos seriam a depuração ou o ajuste de configuração do mesmo ou do sistema, para adequar-se melhor às características do processo executado. Nesse caso, não haveria a necessidade de esperar até o fim da execução para detectar anomalias. Assim que uma dessas fosse detectada o programa poderia ser abortado e ações necessárias para corrigi-las ou amenizá-las poderiam ser realizadas. Um outro uso da apresentação *on-line* dos dados seria a detecção de problemas no sistema, como sobrecarga ou falhas de hardware. Nesse caso, ter-se-ia a comunicação da anomalia quando da sua ocorrência. Isso não seria possível com a obtenção *post-mortem* dos dados.

Em um sistema distribuído, assim como os recursos observados estão distribuídos, os dados de monitoramento também o estão. Portanto, há a necessidade de um mecanismo de coleta, que seja capaz de reunir essas informações de forma que os dados possam ser integrados e possa ser realizada a correlação das informações obtidas.

As transmissões de dados entre dois componentes de um sistema podem ser feitas

apenas quando requisitadas pelo receptor. Esse modelo é chamado *pull* ou sob demanda. No caso contrário, quando o remetente é quem decide quando enviar os dados, sem necessidade de requisição pelo receptor, o modelo é chamado *push*.

O modelo *push* é mais interessante quando se quer observar as variações em um conjunto de características durante um intervalo de tempo. Esse é o caso da análise comportamental de programas e máquinas. Por outro lado, *pull* é mais interessante para sistemas que tem por objetivo descobrir informações sobre o estado atual do sistema em um determinado instante. Um exemplo de tal, seria uma ferramenta utilizada para determinar quais recursos de um sistema estão disponíveis em um determinado instante.

O trabalho aqui apresentado tem por objetivo proporcionar a análise *on-line* do comportamento tanto de sistemas distribuídos quanto da execução de programas nessas plataforma. Para atingir esse objetivo, é introduzido um modelo hierárquico para coleta de dados de monitoramento em sistemas distribuídos. Esse modelo foi integrado ao DIMVisual (SCHNORR; NAVAUX; OLIVEIRA STEIN, 2006), para realização da integração e padronização das informações reunidas. Dessa forma, possibilita-se a sua correlação. O modelo de coleta foi batizado *DIMVisual Hierarquical Collection Model* (DIMVHCM).

Para avaliar o modelo foi implementado um protótipo. Nesse, a hierarquia de coleta fornece dados ao protótipo do DIMVisual, que foi modificado para suportar a integração *on-line* dos dados. Esse protótipo foi utilizado para implementar um módulo de leitura para a ferramenta de visualização TRIVA (SCHNORR; HUARD; NAVAUX, 2008, 2009, 2010; SCHNORR; NAVAUX; HUARD, 2009). Além da leitura dos dados, esse módulo comanda a atualização *on-line* da visualização. Como resultado dessa integração de DIMVHCM, DIMVisual e TRIVA, obteve-se um protótipo de ferramenta de monitoramento *on-line*.

1.1 Monitoramento de Sistemas Distribuídos

O monitoramento de sistemas distribuídos envolve a extração, coleta e interpretação de informações sobre componentes do sistema e sobre processos executados no mesmo e sua apresentação em formato útil para seu usuário (JOYCE et al., 1987). As informações de monitoramento podem ser utilizadas para diversos propósitos, dentre os quais estão (HOLLINGSWORTH; TIERNEY, 2004):

- *Análise de desempenho e ajustes para sua melhoria*: monitoramento pode ser utilizado para observar a execução de aplicações e o comportamento do sistema com o objetivo de encontrar problemas de desempenho. Através da análise dos dados coletados pode-se determinar a causa do baixo desempenho de um programa e analisar alternativas para sua melhoria.

- *Melhorar a próxima execução de um programa*: dados coletados durante a execução de um programa podem ser usados *off-line* para alterá-lo de maneira que o mesmo execute mais rápido na próxima execução.

- *Depuração*: Ferramentas de monitoramento podem ser utilizadas para diminuir a complexidade da depuração de aplicações paralelas, *multithread* e distribuídas.

- *Resolução de problemas e detecção de falhas*: Informações de monitoramento podem ser utilizadas para detectar falhas em um sistema. Adicionalmente, pode haver um mecanismo que realize alguma ação quando uma falha é detectada. Isso pode

variar desde a emissão de alertas para os administradores do sistema até a execução de algum procedimento de recuperação automática de falhas.

- *Tomada de decisões de escalonamento*: um escalonador pode utilizar dados sobre recursos disponíveis no sistema e sobre seu nível de utilização para decidir a quais recursos atribuir a execução de um processo.

- *Adaptação de uma aplicação em execução*: os recursos necessários por uma aplicação podem variar durante sua execução. Dados de desempenho podem ser utilizados para guiar a execução, permitindo a adaptação do comportamento do programa em resposta ao estado dos recursos observados.

- *Auditoria e detecção de invasões*: informações de monitoramento podem ser usadas para atestar o comportamento correto do sistema, ou detectar o uso indevido do mesmo. Além disso, pode-se guardar informações que possibilitem detectar invasores no sistema.

Existem vários problemas associados ao monitoramento de sistemas distribuídos. Um deles é a possibilidade de que os dados gravados não sejam suficientemente atuais. Isso ocorre devido ao atraso causado pela transmissão dos mesmos de onde são coletados até onde são usados. Outro problema é que os eventos não são registrados na ordem de ocorrência, o que torna necessária a utilização de alguma técnica de sincronização dos dados. Além disso, o número de componentes gerando dados de monitoramento pode ser muito grande, dificultando seu gerenciamento e compreensão. Para resolver esse problema é preciso filtrar e processar essa informação. Outra dificuldade é a intrusividade, pois o sistema de monitoramento pode competir pelo uso de recursos com o sistema sendo observado, modificando o comportamento normal do sistema.

Um sistema de monitoramento deve compreender funcionalidades relacionadas à geração de dados de monitoramento, seu processamento, distribuição e apresentação, conforme mostra o diagrama da figura 1.1. A ordem em que as tarefas de cada tipo são realizadas pode variar. Por exemplo, pode ser realizada alguma forma de processamento dos dados tanto antes quanto após sua distribuição, assim como em ambos desses momentos. Tudo isso deve ser projetado e implementado levando em conta os problemas citados anteriormente e também o objetivo do sistema. Esse último é muito importante na tomada de decisão sobre quais aspectos devem ser priorizados no desenvolvimento da ferramenta.

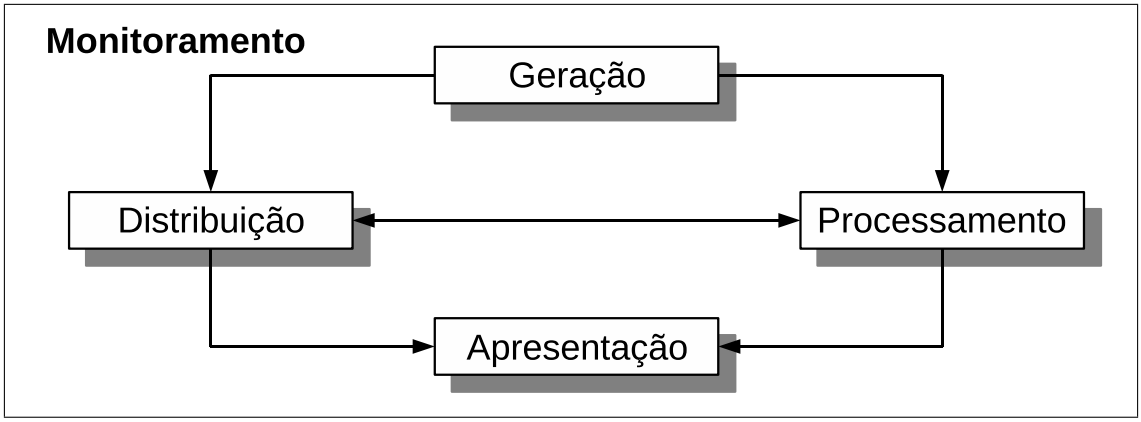


Figura 1.1: Diagrama dos tipos de tarefas realizadas por um sistema de monitoramento.

Como foi dito na introdução deste capítulo, o processo de monitoramento pode ser *post-mortem* ou *on-line*. No primeiro caso, geralmente os dados de monitoramento são armazenados localmente ao recurso monitorado. Após o término do processo, o usuário reúne as informações e utiliza ferramentas para processá-las e apresentá-las. Já no monitoramento *on-line*, a distribuição (ou coleta), processamento e apresentação dos dados são realizados enquanto o fenômeno observado está acontecendo, ou seja, em tempo de execução. Esse método geralmente utiliza um mecanismo de coleta de dados. Dessa maneira, tira-se das mãos do usuário a responsabilidade por essa tarefa.

Uma arquitetura distribuída que vem recebendo bastante destaque hoje em dia é o *Grid* (FOSTER; KESSELMAN; TUECKE, 2001). Esse é um tipo de sistema distribuído de larga escala voltado para o compartilhamento de recursos entre diferentes instituições. O monitoramento de tal sistema deve levar em conta características como grande escala, dinamismo e heterogeneidade dos recursos disponíveis,e distribuição geográfica. Por isso, uma ferramenta de monitoramento de *Grid* deve considerar aspectos específicos em seu projeto. Essas dificuldades tornam interessante seu estudo.

1.2 Apresentação do Trabalho

O objetivo desse trabalho é criar um modelo de coleta de dados de monitoramento que possibilite a análise *on-line* do comportamento de sistemas distribuídos e da execução de programas nesses. Sendo que, neste trabalho, por *on-line* quer-se dizer que os dados devem ser apresentados durante a execução do programa. Não é assumido nenhum compromisso quanto à atualidade das informações. A apresentação das informações é feita através de visualização.

Para possibilitar a apresentação *on-line* dos dados, utiliza-se um mecanismo capaz de obter as informações que estão distribuídas pelos recursos e fornecê-las a um componente capaz de processá-las. Então, é interessante que os dados provenientes de diferentes fontes sejam integrados para possibilitar a visualização conjunta das informações. É necessário ordenar os dados e unificar os identificadores utilizados, pelas diferentes fontes, para os objetos observados. Dessa maneira facilita-se a correlação das informações. Por fim, é desejável que os dados sejam padronizados (convertidos) em um formato que seja útil para o usuário. Esse pode ser uma pessoa ou um programa que utilize os dados para tomar decisões. Um exemplo de padronização seria a conversão das informações para o formato de entrada de uma ferramenta de visualização capaz de mostrá-las de maneira que facilite a compreensão do processo observado.

Para tornar possível a coleta de dados de monitoramento distribuídos, pode-se utilizar um mecanismo hierárquico de coleta de dados distribuídos. Nesse esquema, os dados são obtidos localmente aos recursos, por componentes que os enviam para outros de nível superior. Esses, por sua vez, repassam as informações para níveis mais altos. Assim sucedendo-se até que se tenha todas as informações necessárias em pontos acessíveis por um cliente que as necessite. Essa estrutura torna possível o monitoramento *on-line* (conforme os parâmetros descritos anteriormente), pois possibilita a obtenção das informações enquanto o fenômeno observado está ocorrendo.

O interesse é obter dados que caracterizem uma sequência de eventos que possibilite a análise do comportamento dos objetos observados durante um período de tempo. Subentende-se, portanto, um fluxo contínuo de informações de monitoramento. Por isso é adequado que o envio dos dados seja realizado pelo remetente sem a necessidade de requisição pelo receptor (*push*).

Caso o interesse fosse apenas determinar o estado do sistema em um determinado instante, então um esquema de envio sob demanda (*pull*) poderia adequar-se melhor, pois evitaria envios desnecessários.

Esse trabalho propõe um modelo hierárquico para coleta de dados de monitoramento distribuído. Esse modelo utiliza o modelo *push* de recebimento de dados e tem suporte a um mecanismo de subscrição a informações. Esse mecanismo propõe-se a reduzir o envio de dados inúteis para a análise que está sendo realizada. O modelo de coleta possui três tipos de componentes: coletores, agregadores e clientes.

Os coletores são responsáveis por obter localmente informações geradas por ferramentas de monitoramento ou pela execução de programas instrumentados. Esses dados são então enviados para um conjunto de agregadores.

Os agregadores servem como ponto de acesso a dados recebidos de um conjunto de componentes. Eles formam uma estrutura hierárquica em que um agregador de nível superior recebe dados dos de nível inferior. Esse tipo de componente é responsável por receber dados de monitoramento de coletores ou agregadores de nível inferior e repassá-los a clientes ou agregadores de nível superior.

Os clientes são componentes capazes de receber dados de um conjunto de agregadores e encaminhá-los para um objeto capaz de realizar algum tipo de processamento e análise desses dados.

No presente trabalho, os clientes da estrutura hierárquica de coleta fornecem os dados recebido para fontes de dados do DIMVisual (SCHNORR; NAVAUX; OLIVEIRA STEIN, 2006). Esse é um modelo de integração de dados de monitoramento que recebe dados de diferentes fontes (ferramentas e máquinas), os integra e padroniza. O resultado são dados unificados, no formato de entrada de uma ferramenta de visualização.

Os dados gerados pelo DIMVisual são recebidos por um componente que os utiliza como entrada para uma ferramenta de visualização. Nesse trabalho utilizou-se técnicas de visualização cujo uso na análise do comportamento de aplicações paralelas foi proposto por Schnorr et al (SCHNORR; HUARD; NAVAUX, 2008, 2009, 2010; SCHNORR; NAVAUX; HUARD, 2009). Isso foi feito através do uso de um protótipo de ferramenta de visualização, chamado TRIVA, que implementa as técnicas e foi desenvolvido por seus autores. Essas técnicas compreendem o uso de *treemaps* e de visualização 3D para mostrar a evolução temporal de aplicações.

Para avaliar o modelo foi implementado um protótipo. Inicialmente realizou-se a implementação de classes que possibilitam a construção de coletores, agregadores e clientes. Logo após foram implementadas versões de teste desses componentes. Na etapa seguinte, o cliente foi modificado para fornecer dados para o protótipo do DIMVisual e esse foi modificado para suportar a integração *on-line* dos dados. Como uma última etapa da implementação do protótipo, foi feita a ligação do DIMVisual com a ferramenta de visualização, de maneira a tornar esta também *on-line*.

Para isso foi desenvolvido um módulo para o TRIVA, chamado *DIMVHCM Reader*. Esse utiliza o protótipo do DIMVisual para receber dados de monitoramento integrados, no formato Pajé e fornecê-los ao TRIVA. Além disso, foi necessário implementar a atualização da visualização conforme novos dados tornam-se disponíveis. Essa atualização não era feita anteriormente devido ao protótipo suportar apenas a visualização *post-mortem* das informações.

Utilizando o protótipo realizou-se experimentos para avaliar o tempo necessário para o envio de dados de monitoramento. Essa avaliação obteve resultados promissores nos casos testados. Também foi avaliado o desempenho do cliente implementado. Esses

testes avaliaram a capacidade do cliente processar os dados recebidos.

1.3 Organização do texto

O próximo capítulo trata do monitoramento de sistemas distribuídos. Nele serão abordadas as etapas envolvidas nessa tarefa, assim como algumas dificuldades enfrentadas no processo. Também são discutidas formas de representação das informações de monitoramento. Além disso, na seção 2.4 são apresentadas características do monitoramento de *Grid*. Também são apresentadas a *Grid Monitoring Architecture* (GMA) e uma taxonomia para classificação de sistemas de monitoramento de Grid. No capítulo 3 são apresentados o DIMVisual e o TRIVA e também um estudo de um conjunto de sistemas de monitoramento. Depois disso, no capítulo 4, é apresentado o modelo de coleta de dados de monitoramento proposto no presente trabalho. Nesse capítulo há também uma comparação do modelo com as ferramentas estudadas no capítulo anterior. O capítulo 5 aborda a implementação do protótipo utilizado para avaliar o modelo. Essa avaliação é apresentada no capítulo 6. Por fim, no capítulo 7, tem-se a conclusão dessa dissertação. Lá também são apresentadas possibilidades a serem exploradas futuramente, dando continuidade a esse trabalho.

1.4 Considerações Finais

Há uma grande variedade de áreas em que pode-se utilizar monitoramento. Uma dessas é a análise do comportamento de sistemas e programas distribuídos. Para isso é necessário reunir as informações espalhadas pelo sistema, integrá-las e padronizá-las.

Esse trabalho propõe um modelo hierárquico para coleta de dados de monitoramento. Seu objetivo é possibilitar a análise *on-line* do comportamento de sistemas e programas utilizando visualização. Esse modelo utiliza o modelo *push* de recebimento de dados, para fornecê-los assim que se tornam disponíveis. Além disso, ele inclui um mecanismo de subscrição que proporciona a redução do envio de informações desnecessárias.

Quanto à sua estrutura, o modelo é composto por coletores, agregadores e clientes. Os coletores obtêm os dados de monitoramento localmente e os transmitem a agregadores. Os agregadores formam uma hierarquia, através da qual os dados são retransmitidos até os clientes. Os clientes recebem os dados da hierarquia de agregadores e os encaminham para componentes que farão o seu processamento.

Foi implementado um protótipo que fornece dados para o DIMVisual, que é um modelo de integração de dados. Os dados integrados, por sua vez, são usados como entrada para a ferramenta de visualização TRIVA. Experimentos foram realizados, para analisar o desempenho da transmissão dos dados pelo modelo de coleta. Além disso, foi testada a capacidade de processamento de dados pelo cliente.

2 MONITORAMENTO DE SISTEMAS DISTRIBUÍDOS

O objetivo desse capítulo é apresentar uma visão geral do processo de monitoramento de sistemas distribuídos. Funcionalidades envolvidas na geração, distribuição, processamento e apresentação das informações de monitoramento são apresentadas. Também são abordados aspectos relacionados à implementação de ferramentas de monitoramento de sistemas distribuídos. Um desses é a instrumentação e sua relação com a intrusividade do monitoramento. Outros são a obtenção de um tempo global consistente e a ordenação de eventos. Além disso, são apresentadas algumas formas de representação dos dados: escalares, rastros e vetores temporais.

Adicionalmente, na seção 2.4, são abordados aspectos relacionados ao monitoramento de *Grid*. Isso inclui a apresentação da *Grid Monitoring Architecture* (GMA). Além disso, é apresentada uma taxonomia de ferramentas de monitoramento de *Grid*.

2.1 Etapas do Monitoramento de Sistemas Distribuídos

O monitoramento de sistemas distribuídos compreende quatro etapas: a geração dos dados de monitoramento; o processamento desses dados; sua distribuição; e finalmente, sua apresentação. Cada uma dessas etapas pode envolver várias tarefas, que podem ser realizadas de diferentes maneiras. Esses aspectos serão abordados nas quatro subseções a seguir, uma para cada etapa citada.

2.1.1 Geração dos Dados de Monitoramento

A geração de dados de monitoramento compreende tarefas como a obtenção de informações de estado, a geração de notificações de eventos e a formação de rastros de monitoramento. A figura 2.1 mostra essas tarefas em um diagrama, juntamente com algumas alternativas relacionadas a seu funcionamento, as quais serão abordadas mais adiante nessa seção.

Cada componente do sistema a ser monitorado possui um estado, que pode ser representado por uma série de valores que o caracterizam em um determinado instante. Adicionalmente ao seu estado, cada componente possui eventos associados. Esses refletem mudanças de estado, tornam-nas observáveis. Dados gerados através do uso das informações de estado de um componente em um determinado instante podem ser obtidos de duas maneiras:

- *Relatórios periódicos*: os dados são enviados sempre que se completa um período de tempo.
- *Sob demanda*: é utilizado algum mecanismo que permita aos interessados nas infor-

mações de estado solicitar seu recebimento quando necessitarem. Nesse caso, um mecanismo de *polling* também pode ser usado, caso se queira receber informações periódicas.

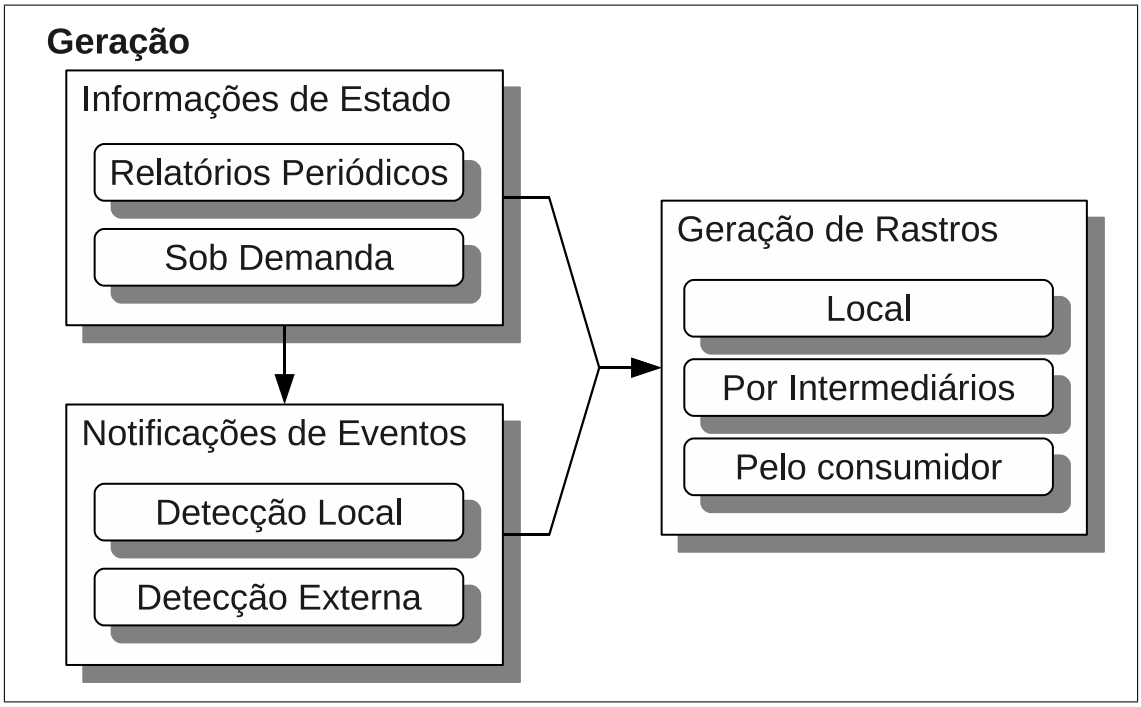


Figura 2.1: Diagrama das tarefas relacionadas à geração de dados dados de monitoramento

O uso de eventos possibilita o registro de mudanças no estado dos componentes monitorados. Um evento é gerado quando um conjunto de condições pré-definidas é satisfeito. Para detectar a ocorrência de eventos, sensores devem ser adicionados aos objetos ao qual o mesmo refere-se. Esse processo é chamado instrumentação. Para decidir onde, como e quando realizar a detecção de eventos, devem ser usados como critérios os recursos disponíveis para isso e a intrusividade do sistema de monitoramento.

Quanto à localização, os eventos podem ser detectados internamente ou externamente ao componente monitorado. No primeiro caso, poderia ser realizada uma checagem logo após a modificação de informações de estado. No segundo, um componente externo, que recebe informações de estado do objeto monitorado, poderia realizar a detecção de eventos.

Quanto ao tempo, a detecção de eventos pode ser imediata ou postergada. Por exemplo, no caso do uso de um monitor em *hardware*, as mudanças de estados poderiam ser detectadas imediatamente, através de sinais em um barramento. Em outro caso, informações de estado fornecidas pelo componente monitorado poderiam ser armazenadas e posteriormente examinadas, para detecção de eventos.

Quando um evento é detectado é gerada uma notificação. Essa contém uma série de atributos de interesse, como identificadores, *timestamps*, tipo e valores relacionados ao estado do objeto. Alternativamente pode-se ter uma geração de notificações em mais de um estágio. Por exemplo, o objeto monitorado poderia gerar notificações mínimas e um outro componente, que receberia a notificação, geraria uma mais complexa, adicionando dados como *timestamps* e outros tipos de atributos. A quantidade de informação contida em uma notificação depende das necessidades dos clientes que as utilizam.

Para a realização de algumas tarefas, dados que descrevam o comportamento de um conjunto de componentes durante um período de tempo podem ser desejáveis. Com esse objetivo, eventos e informações de estado podem ser armazenados em ordem de ocorrência, formando rastros de monitoramento. Exemplos de usos desses rastros incluem (Mansouri-Samani and Morris Sloman, 1995):

- *Arquivamento e análise post-mortem*: Os dados podem ser arquivados para posterior processamento, análise e utilização. Isso pode ser interessante para propósitos de depuração e análise de desempenho, em que pode-se efetuar a análise *post-mortem* das informações. Dessa forma, diminui-se a intrusividade da monitoração, através da postergação da análise dos dados para após o fim da execução das tarefas que estão sendo observadas.

- *Escassês de Recursos Disponíveis*: pode ser que em um determinado sistema não existam recursos computacionais suficientes para a análise e interpretação das informações. Além disso, os recursos comunicacionais, através dos quais os dados são enviados para um componente de processamento externo, podem ser limitados. Nesse caso, os rastros armazenados podem ser utilizados para processamento posterior. Caso não fossem gravados esses rastros, provavelmente parte dos dados gerados teriam que ser descartados por não haver recursos para processá-los no momento em que se tornam disponíveis.

- *Velocidade da Visualização*: Quando a frequência de recebimento dos dados de monitoramento é muito rápida, pode ser difícil seu acompanhamento por um observador. O uso de rastros permite que o observador controle a velocidade da visualização e também possa avançar e voltar na mesma, podendo analisar eventos que aconteceram em diferentes instantes.

- *Transformação da Visão Lógica da Atividade do Sistema*: é possível a construção de rastros de monitoramento globais a partir de rastros locais. Também é possível a geração de diferentes rastros, que reflitam uma visão especializada ou restrita do sistema, a partir de um único rastro.

Para cada evento registrado, deve ser gravado um *timestamp*, um identificador de seu tipo, um identificador do componente a que se refere e um ou mais valores medidos. Algumas alternativas devem ser levadas em conta no projeto de mecanismos de geração de rastros, entre elas:

- *Local onde os rastros são gerados*: os rastros podem ser formados ou pelos objetos que originam os dados de monitoramento, ou por componentes intermediários de monitoramento ou pelo usuário final da informação.

- *Rastros temporários ou persistentes*: rastros podem ser mantidos em um *buffer* temporário. Geralmente, quando se usa essa estratégia, os dados são descartados após serem utilizados. Alternativamente, os rastros podem ser gravados em armazenamento persistente. Nesse caso, os dados usualmente não são apagados quando lidos.

- *Capacidade de armazenamento*: Há a necessidade de adotar-se uma estratégia para quando o tamanho do rastro atinge a capacidade de armazenamento disponível. Entre essas pode-se citar a sobrescrita de rastros mais antigos, e o descarte de novos

rastros. Outra estratégia que, embora não resolva o problema, dificulta seu surgimento é a compressão dos dados mais antigos através da sumarização dos mesmos. Pode-se, por exemplo, guardar apenas as médias dos valores numéricos menos recentes.

- *Sofisticação da geração dos rastros*: um mecanismo simples seria o armazenamento dos dados em um *buffer* de rastros local ao objeto monitorado, em ordem de chegada, sem realizar qualquer modificação dos mesmos. Um mecanismo sofisticado seria o uso de um serviço de registro de eventos para a criação de rastros persistentes. Isso poderia ser acompanhado de geração de múltiplos rastros, proporcionando diferentes visões lógicas da atividade do sistema. Para essa tarefa utilizar-se-ia filtragem para selecionar os dados que fariam parte de cada rastro.

- *Acesso sob demanda aos rastros*: pode ser interessante que componentes que realizam processamento dos rastros possam acessá-los sob demanda. Dessa forma, cabe a esses componentes decidir o momento mais oportuno para a realização dessa tarefa.

- *Condições pré-determinadas para transferência de informações*: podem ser definidas condições para determinar quando as informações de monitoramento serão transmitidas. Dessa maneira pode-se estipular, por exemplo, que os dados só sejam transmitidos quando o *buffer* está cheio, ou haja pelo menos *n* elementos no mesmo e a carga do meio de comunicação esteja abaixo de um limite estipulado.

2.1.2 Processamento de Dados de Monitoramento

Uma vez gerados os dados de monitoramento, algumas tarefas de processamento são realizadas sobre eles. Isso pode compreender, como mostra o diagrama da figura 2.2, a integração de múltiplos rastros, a geração de rastros específicos a partir de um mais abrangente, a validação dos dados, sua filtragem, a combinação ou correlação dos mesmos e a sua análise. Essas tarefas são o assunto do restante dessa subseção.

Rastros de monitoramento podem ser organizados e formados de várias maneiras. Isso possibilita o fornecimento de várias visões lógicas do sistema. A construção desses rastros é feita de acordo com critérios relacionados às informações fornecidas por eles. Esses critérios especificam como formar um rastro e o que ele conterá. Rastros de monitoramento são gerados através do processamento direto de eventos e informações de estado ou a partir de rastros preexistentes.

Rastros podem ser gerados a partir da integração de vários outros rastros, que contenham informações mais específicas. Por exemplo, rastros com informações locais podem ser utilizados para construir rastros globais. Para a realização da integração, os rastros originais são percorridos, selecionando-se as informações que devem ser incluídas no rastro resultante. Esse deverá ser composto pelos dados selecionados gravados de forma ordenada. Um exemplo de trabalho que utiliza a integração de rastros é o DIMVisual (SCHNORR; NAVAUX; OLIVEIRA STEIN, 2006). Esse integra rastros de diferentes origens e gerados por diversas ferramentas, os unifica e padroniza, gerando uma visualização conjunta dos mesmos.

Outra forma de processamento, que pode ser aplicada a rastros de monitoramento, é a geração de rastros mais específicos a partir de um mais abrangente. Isso permite que se tenha diferentes visões do sistema, cada uma com um nível de detalhamento ou contendo apenas as informações necessárias ao usuário dos rastros. Isso também pode ser utilizado

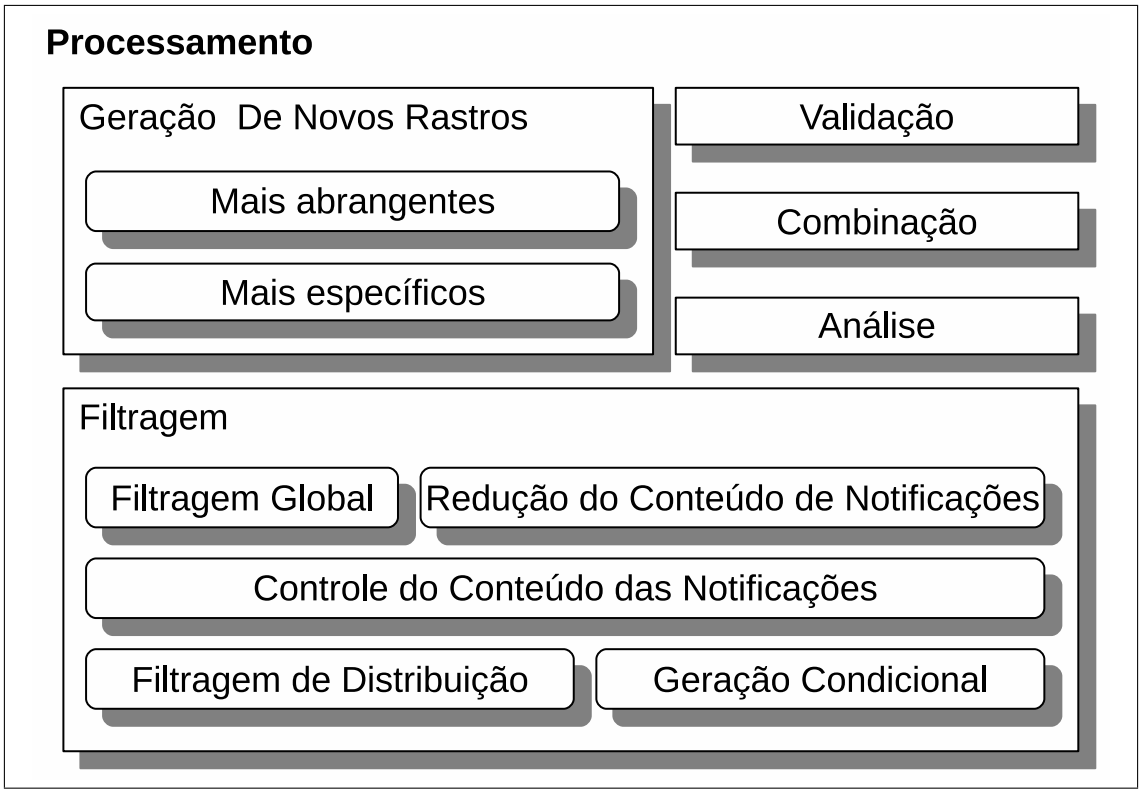


Figura 2.2: Diagrama das tarefas relacionadas ao processamento de dados de monitoramento

na segurança do sistema, fornecendo ao usuário rastros contendo apenas as informações a que ele tem acesso permitido.

Um outro tipo de processamento é o emprego, por alguns sistemas de monitoramento, de técnicas para verificar a validade dos dados. Essas incluem, entre outras, a verificação da validade de identificadores, ou do *timestamp* dos dados. O objetivo, nesse caso, é aumentar a certeza de que os dados estão corretos.

Uma outra tarefa de processamento seria o armazenamento de dados de monitoramento, para criação de um modelo do estado corrente do sistema. Esses dados poderiam ser usados por outros usuários, gerentes e agentes de processamento. Em sistemas distribuídos, no entanto, vários fatores impossibilitam manter dados realmente atualizados e consistentes, que representem o estado atual do sistema. Entre esses estão atrasos de comunicação e falhas de componentes.

Uma maneira de tornar as informações de monitoramento mais compreensíveis e úteis para seus usuários é a combinação, também chamada correlação, dos dados. Essa técnica consiste em aumentar o nível de abstração dos dados de monitoramento. Esse processo é realizado utilizando definições que especificam novos eventos e informações de estado baseados em combinações de outros. Ou seja, dados em um nível de abstração são utilizados para gerar dados em um nível superior. Por exemplo, pode-se diminuir a granularidade dos dados com relação ao tempo, através do cálculo das médias dos valores medidos durante intervalos periódicos. Outra possibilidade é a apresentação de dados sobre um conjunto de objetos monitorados, gerados a partir de valores obtidos de seus membros. Um exemplo disso seria o cálculo da ocupação do sistema a partir das medidas da ocupação dos seus nós. Nesse caso, poder-se-ia ter a média das cargas dos nós, a taxa de utilização relativa à capacidade total de processamento disponível ou até informações

geradas através do uso de técnicas mais complexas, levando em conta outras variáveis e características do sistema.

O grande volume de dados de monitoramento gerado tipicamente por um sistema distribuído causa um alto uso de CPU e banda de comunicação. Além disso, os usuários desses dados podem ser incapazes de compreendê-los devido à sobrecarga de informações. Para resolver esse problema, a quantidade de dados recebida pelos usuários deve ser limitada a um nível de detalhamento relevante para seus objetivos. O processo de minimização desses dados recebe o nome de filtragem. Esse mecanismo também é necessário para a segurança, pois alguns usuários podem não ter direito de acesso a parte das informações de monitoramento. A filtragem pode ser realizada em diferentes estágios e locais:

- *Filtragem Global*: dados que não satisfazem um critério global de filtragem são descartados.
- *Redução do Conteúdo das Notificações*: um objeto que recebe uma notificação pode gerar um nova contendo apenas um subconjunto das informações recebidas.
- *Controle do Conteúdos das Notificações*: critérios são usados na fase de geração de dados de maneira que apenas informações necessárias são inclusas em cada notificação.
- *Geração Condicional*: dados são gerados somente quando condições pré-definidas são satisfeitas. Por exemplo, um valor medido, ou a variação desse, deve ser maior que um limite pré-definido, ou um período de tempo determinado deve haver decorrido desde a última notificação. Além disso, pode ser possível ativar e desativar os mecanismos de geração de notificações.
- *Filtragem de Distribuição*: a distribuição de notificações baseada em um esquema *publish / subscribe* fornece filtragem implícita dos dados. Isso ocorre devido a, nesse caso, notificações serem enviadas somente aos componentes que requisitaram inscrição para recebê-las. Um rastro de monitoramento diferente pode ser gerado para cada objeto de destino.

O melhor momento para a realização de filtragem é nos estágios inicias. Dessa forma, reduz-se o uso de recursos nos estágios seguintes. No entanto, principalmente caso seja realizada no componente monitorado, deve-se ter atenção especial no consumo de recursos. Isso é muito importante, pois se a quantia de recursos necessária para a filtragem for alta, ela pode gerar um nível de intrusividade indesejável.

A análise das informações de monitoramento pode ser feita de diversas maneiras, de acordo com o objetivo pretendido. Alguns tipos de análise incluem o cálculo de médias e variâncias de determinadas variáveis, análise de tendências para realização de previsões, como feito pela ferramenta *Network Weather Service* (WOLSKI; SPRING; HAYES, 1999), e correlação de eventos para diagnóstico de falhas. A apresentação das informações também inclui aspectos de análise, por exemplo, a geração de representações gráficas do comportamento do sistema. Outro tipo de análise pode ser feita com o objetivo de adaptar a execução de programas conforme o estado dos recursos disponíveis. Autopilot (RIBLER et al., 1998) é um exemplo de ferramenta que realiza esse tipo de processamento. Por fim, a análise pode variar de uma coleta simples de estatísticas até análises sofisticadas baseadas em modelos.

2.1.3 Distribuição de Informações de Monitoramento

Os dados de monitoramento gerados pelos componentes do sistema tem de ser transmitidos para seus usuários. Métodos de distribuição dessas informações variam desde esquemas simples e fixos até bastante complexos e especializados. Um mecanismo simples poderia ser o *broadcast* das mesmas para todos os componentes. Um esquema mais complexo e especializado poderia ser baseado no princípio *publish / subscribe*. Em um mecanismo desse tipo, clientes inscrevem-se para receber as informações de estado e eventos de que necessitam. Essas inscrições são registradas em um serviço que as gerencia. Dessa forma, pode-se determinar quais dados devem ser enviados para quais clientes, o que provê filtragem implícita dos dados.

Além disso, pode ser usado algum tipo de estrutura que possibilite a obtenção de informações de diferentes conjuntos de objetos, possivelmente não interconectados diretamente. Por exemplo, nós de um *cluster* podem não ter acesso direto aos de outros *clusters*. Uma solução para esse problema é a utilização de um componente republicador, que forneça acesso a dados sobre um conjunto de objetos. Por exemplo, a ferramenta NetLogger (GUNTER et al., 2000) permite a especificação de um nó para onde as informações de monitoramento devem ser enviadas. No entanto, isso implica que todos os objetos monitorados necessitam ter acesso a esse nó. Em uma versão mais avançada dessa ferramenta (GUNTER et al., 2003), foram adicionados produtores locais que fornecem acesso a informações sobre um conjunto de objetos. Esses registram-se em um serviço a partir do qual podem ser descobertos por clientes que desejem acessá-los. Essa nova versão também suporta o modelo *publish / subscribe* para recebimento de informações de monitoramento de um produtor.

Uma outra solução é fazer com que os dados fornecidos pelos republicadores possam ser coletados por outros republicadores, formando uma hierarquia. Dessa forma possibilita-se o acesso a informações sobre vários conjuntos de objetos a partir de um único ponto. Um exemplo de sistema de distribuição que utiliza uma estrutura hierárquica para agregar dados de diferentes *clusters* é a ferramenta Ganglia (MASSIE; CHUN; CULLER, 2004), tratada com mais detalhes na seção 3.3.2.

2.1.4 Apresentação de Informações de Monitoramento

Após feito todo o processo de geração, coleta e processamento das informações de monitoramento, essas devem ser apresentadas aos clientes. Os dados devem estar em formato adequado para que os mesmos possam ser utilizados na realização da tarefa para que foram projetados. Uma interface de usuário adequada deve permitir ao usuário especificar como mostrar a informação. Além disso deve saber lidar com o grande volume de dados gerado pelo sistema, informações em vários níveis de abstração, o paralelismo das atividades dos sistema e a velocidade com que essa informação é produzida e apresentada.

Várias estratégias podem ser utilizadas para a exibição de informações de monitoramento ao usuário. A mais comum delas é a textual, que pode utilizar indentação, destaque e coloração para aumentar a expressividade da exibição. No entanto, essa técnica não é adequada à apresentação de atividades em paralelo (Mansouri-Samani and Morris Sloman, 1995).

Outra estratégia é o uso de diagramas processo-tempo que representam em um diagrama bidimensional o estado de sistemas paralelos. Um dos eixos do diagrama representa os componentes monitorados, o outro representa o tempo. Ele mostra o estado atual do sistema e os eventos que o levaram a esse estado. Gráficos podem ser utilizados para

esse tipo de representação. Nesse caso, há a possibilidade de representar-se interações entre componentes, também. Um exemplo de exibição desse tipo pode ser gerado usando a ferramenta Pajé (KERGOMMEAUX; OLIVEIRA STEIN, 2000), utilizando um diagrama de Gantt para mostrar as mudanças de estado com o passar do tempo e setas ou linhas para representar interações entre componentes.

Além disso pode-se utilizar animação na exibição, permitindo a observação instantânea do estado do sistema. Nesse caso, os componentes seriam representados na tela por objetos que utilizam algum tipo de animação para representar visualmente seu estado atual e a mudança do mesmo. Essa animação pode ser feita utilizando gráficos, ícones, listas de estados, e várias outras formas de representação.

Com relação à interface do usuário, pode-se enumerar uma série de características desejáveis, como a visualização em vários níveis de abstração, o posicionamento dos dados de monitoramento, o controle da exibição com relação ao tempo, o uso de múltiplas visualizações, e a visibilidade de dados sobre interações.

Quanto à primeira dessas características, é interessante que a interface possibilite ao usuário observar o comportamento do sistema no nível de abstração desejado. Ele deve ser capaz de iniciar a observação em um nível mais alto e progressivamente focar em níveis mais baixos. Além disso, é importante que o usuário seja capaz de focar-se em informações que sejam de importância imediata, sem informações excedentes poluindo a visualização. Para conseguir isso pode-se focar em um certo nível de abstração e utilizar técnicas combinação e filtragem para mostrar apenas as informações interessantes.

Outra característica desejável da interface é que ela permita a reorganização espacial das informações. Isso permite, por exemplo, que o usuário posicione informações que quer examinar em conjunto próximas umas das outras. Também é desejável que o usuário possa percorrer as informações com relação ao tempo. Ou seja, o usuário poderia rever informações, através de dados históricos, podendo voltar e avançar a visualização em relação ao tempo.

Além disso, a interface deveria possibilitar o uso de diferentes visões do sistema. Isso corresponde a poder mostrar o comportamento do sistema a partir de diferentes pontos de vista, por exemplo, em diferentes níveis de abstração ou mostrando informações de diferentes conjuntos de componentes ou diferentes tipos. Essa funcionalidade facilita a correlação dos dados pelo usuário. Outra funcionalidade desejável é que se possa visualizar interações entre os componentes monitorados. Exemplos dessas seriam trocas de mensagens e acessos a variáveis compartilhadas. Também pode ser interessante que se tenha acesso a informações de nível mais baixo, como o conteúdo das informações de estado e das notificações de eventos.

2.2 Aspectos Relacionados à Implementação

Na implementação de sistemas de monitoramento é necessário preocupar-se com algumas dificuldades. Uma das principais é a intrusividade do sistema de monitoramento. Outra, é a inexistência de um tempo global sincronizado em sistemas distribuídos. A subseção a seguir trata da intrusividade e de sua relação com o tipo de monitor utilizado, assim como apresenta esses tipos. Logo após, a subseção 2.2.2 aborda o problema da obtenção de um estado global do sistema e aspectos relacionados à temporização e ordenação de eventos.

2.2.1 Intrusividade do Sistema de Monitoramento

Intrusividade é o efeito que o monitoramento tem sobre o comportamento do sistema monitorado. Ela acontece devido ao sistema de monitoramento compartilhar recursos com o mesmo. Monitores intrusivos podem alterar arbitrariamente a datação de eventos no sistema e podem levar à degradação do desempenho desse, mudança da ordenação global dos eventos, resultados incorretos, aumento do tempo de execução da aplicação e mascaramento ou criação de *deadlocks*.

A maneira como o sistema de monitoramento detecta eventos é uma característica que influencia bastante a intrusividade do sistema. Existem diversos mecanismos que podem ser utilizados para essa detecção, os quais podem ser caracterizados em três tipos: monitores em *hardware*, em *software* e híbridos.

Um monitor em *hardware* consiste de um componente separado que detecta eventos de outros componentes. Ele pode utilizar-se da observação de barramentos do sistema, sensores físicos conectados aos processadores, portas de memória, entre outros. A vantagem desse monitor é a não intrusividade, devido ao não compartilhamento de recursos com o sistema monitorado. As desvantagens desse tipo de monitor são:

- O custo advindo da necessidade de *hardware* adicional.
- O baixo nível dos dados fornecidos não satisfaz os requisitos dos programadores de aplicações em ambientes paralelos. Adicionalmente, o custo de processamento e complexidade dos mecanismos necessários para prover monitoramento em nível de aplicações a partir desses dados é muito alto.
- Esses monitores formam a classe menos portável de mecanismos de monitoramento.

Monitores em *software* geralmente compartilham recursos com o sistema monitorado. Programas são instrumentados inserindo pontos de amostragem no código, para a detecção de eventos. Esses monitores têm as vantagens a seguir:

- Informações apresentadas de uma maneira orientada à aplicação, o que faz com que sejam mais fáceis de entender e utilizar, se comparadas a informações de baixo nível.
- São portáveis e facilmente replicáveis.
- São mais fáceis de projetar e implementar e mais flexíveis que os monitores em *hardware*.
- São mais baratos que monitores em *hardware* devido à desnecessidade de recursos de *hardware* dedicado adicionais.

A desvantagem desses monitores é o compartilhamento de recursos com o sistema observado, interferindo no mesmo e modificando, assim, seu comportamento. Esse problema aumenta se os dados forem processados e exibidos *on-line*. Existem várias abordagens quanto a onde realizar a instrumentação de *software*. Pode-se instrumentar o código fonte, bibliotecas, o código objeto e o *kernel*.

A instrumentação do código fonte consiste em inserir pontos de amostragem no código fonte do programa. A desvantagem dessa técnica é a necessidade de recompilar o

código a cada mudança na instrumentação. Os pontos de amostragem podem ser inseridos manualmente ou automaticamente. A instrumentação manual é difícil, consome tempo e tende ao erro. Com programas paralelos essa tarefa torna-se ainda mais complexa. A instrumentação pode ser feita utilizando ferramentas como a NetLogger (GUNTER et al., 2000), que fornece uma API e uma biblioteca com funções para simplificar essa tarefa.

Uma alternativa à instrumentação direta do código fonte é a utilização de bibliotecas instrumentadas. Uma vantagem desse método é o alto nível de portabilidade. Outra é que, quando o monitoramento não for necessário, o programa pode ser ligado a uma versão não instrumentada da biblioteca. A desvantagem dessa alternativa é que somente eventos para os quais há instrumentação na biblioteca podem ser detectados. Para minimizar esse problema, pode ser fornecida uma função especial de rastreamento que possa ser inserida no código fonte pelo programador, para detectar um determinado evento.

Um terceiro local onde a instrumentação pode ser inserida é o código objeto. Isso pode ser feito através da utilização de um compilador especial que gere código instrumentado. Ao invés de código objeto, esse compilador poderia gerar um código intermediário, independente de máquina, o que aumentaria a portabilidade do código instrumentado. A desvantagem dessa abordagem é a necessidade de um compilador especial. Uma outra possibilidade é a instrumentação dinâmica do código, durante a execução do programa, como feito pela ferramenta Paradyn (MILLER et al., 1995). Essa realiza a instrumentação durante a execução do programa.

Outra possível abordagem seria a instrumentação do *kernel* do sistema operacional para detecção de eventos do sistema. A vantagem disso é que a detecção de eventos fica transparente para a aplicação. Uma desvantagem é que apenas eventos relacionados a chamadas do sistema podem ser detectados. Similarmente ao que acontece com a instrumentação de bibliotecas, uma função especial de rastreamento pode ser adicionada ao *kernel*. Essa função seria chamada quando fosse necessário registrar um evento de aplicação. Dproc (JANCIC et al., 2002; AGARWALA et al., 2003) e *Fast Kernel Tracing* (RUSSELL; CHAVAN, 2001) são exemplos de ferramentas baseadas na instrumentação do *kernel* do sistema operacional.

Monitores híbridos tentam beneficiar-se das vantagens tanto de monitores em *hardware* quanto em *software*. Eles possuem recursos independentes mas também compartilham alguns recursos com o sistema observado. Geralmente há um dispositivo de *hardware* independente que recebe informações de monitoramento de programas instrumentados. As notificações de eventos são geradas, processadas e exibidas por *hardware* dedicado.

A principal vantagem desses monitores é a menor intrusividade, se comparados a monitores implementados puramente em *software*. Também, ao contrário dos monitores em *hardware*, eles geram informações de monitoramento de alto nível, orientadas a aplicações. Adicionalmente, eles são mais intrusivos que monitores puramente em *hardware*, mas são mais baratos que esses por necessitarem menos recursos dedicados. Além disso, monitores híbridos são menos portáveis que os em *software*, devido ao *hardware* dedicado utilizado.

2.2.2 Estado Global, Tempo e Ordenação de Eventos

Um sistema distribuído tipicamente é formado por um conjunto de nós independentes que cooperam entre si e comunicam-se através de mensagens, sem memória compartilhada ou um relógio comum. Esses sistemas são mais difíceis de monitorar que sistemas centralizados. Isso se deve ao paralelismo entre os processadores e nós, atrasos de comunicação aleatórios e não negligenciáveis, falhas parciais e inexistência de um tempo

global sincronizado.

Devido a essas características, diferentes execuções de um mesmo algoritmo, em um desses sistemas, podem resultar em diferentes intercalações de eventos. Isso faz com que o comportamento do sistema seja não-determinístico e imprevisível. Além disso, por causa dos atrasos arbitrários na transmissão de mensagens, é impossível obter um *snapshot* instantâneo e consistente do sistema. Outro problema é que a falta de um tempo global dificulta a determinação de relações de causa-efeito entre eventos através da análise de rastros de monitoramento.

Existem alguns mecanismos utilizados para tentar obter uma sincronização física ou lógica de relógios. Um deles é a sincronização física de relógios que consiste no envio de mensagens contendo *timestamps*. Esses *timestamps* podem provir de um relógio externo de alta precisão ou podem ser locais. Caso em que os nós utilizam algum algoritmo para ajustar seus relógios, tentando mantê-los dentro de um desvio máximo de tempo entre si. Uma desvantagem dessas técnicas é que elas introduzem um sobrecusto no sistema monitorado.

Outra maneira de tentar manter um tempo global no sistema é o uso de relógios lógicos, que são baseados na relação de causa, e estabelecem uma ordenação parcial dos eventos no sistema monitorado (LAMPORT, 1978). Essa técnica consiste primeiramente em avançar o relógio lógico entre quaisquer dois eventos. Além disso, ao receber uma mensagem, caso o *timestamp* dessa for maior ou igual ao seu, um processo deve avançar seu relógio para depois desse *timestamp*. Algumas vantagens desse mecanismo são o baixo sobrecusto e a pequena adição de informação nas mensagens.

Outra alternativa é o uso de um vetor de *timestamps* lógicos, contendo um elemento para cada processo. Cada processo incrementa seu próprio elemento e envia o vetor junto com suas mensagens. Ao receber uma mensagem o processo atualiza seu vetor com os dados recebidos de outros processos. Um problema dessa abordagem é o custo de manter um elemento para cada processo e também de incluí-lo em cada mensagem.

Outra técnica envolve a construção de um *snapshot* global da atividade do sistema (CHANDY; LAMPORT, 1985). Esse mecanismo envolve o sistema inteiro na coleta de informações de estado e utiliza o envio de marcadores através de todos os *links* de cada processo, o que o torna altamente ineficiente quando se quer informações no escopo de apenas um pequeno subconjunto dos componentes do sistema.

2.3 Representação dos Dados de Monitoramento

O formato como os dados de monitoramento são representados é um importante fator que influencia a eficiência da instrumentação, o espaço de armazenamento necessário e os tipos de análises que podem ser realizados. Os dados podem ser representados por estruturas de tamanhos finito ou por fluxos de rastro. O primeiro tipo de representação compreende contadores e temporizadores, associados à estrutura dos programas. Exemplos desses são contadores de instruções e de procedimentos. Esses são atualizados em pontos estratégicos dos programas. No caso dos rastros, um conjunto de dados é gerado cada vez que o programa executa uma instrução rastreada.

2.3.1 Contadores e Temporizadores

Um contador é o tipo mais simples de dado de desempenho. Seu valor é incrementado em pontos apropriados da execução do programa. Esses podem estar associados a eventos de monitoramento de *software* ou *hardware*. O principal aspecto a ser levado em

conta com relação a contadores é o tamanho. 32 *bits* são suficientes para a maioria das operações de *software*. No entanto, à medida que o monitoramento aproxima-se do nível do processador a frequência de atualização do contador aumenta. Por isso, podem ser necessárias variáveis de maior tamanho, que não são suportadas por muitas arquiteturas. Assim, pode ser necessário utilizar uma sequência de variáveis de 32 *bits* para armazenar os valores dos contadores.

Temporizadores são altamente complexos de serem implementados. As principais preocupações dizem respeito à precisão e velocidade de acesso. Processadores modernos possuem contadores em *hardware* para prover tempos de execução e absolutos. O tempo absoluto é importante para medidas de desempenho, contabilizando o tempo em que o processo está bloqueado e não apenas o tempo de execução. Normalmente o acesso ao tempo em termos do número de ciclos do processador é rápido. Por outro lado, dados temporais com medidas de grão maior (em milissegundos, por exemplo), são mais lentos de serem obtidos.

2.3.2 Rastros

O tipo mais geral de dado de monitoramento é o rastro. Dados são adicionados a um rastro em pontos selecionados da execução do programa. Esses dados podem ser compostos por qualquer tipo de informação. Mas geralmente há duas seções: um cabeçalho e dados dependentes do tipo de evento. O cabeçalho contém informações comuns a todos eventos, como tipo, *timestamp* e tamanho. Além disso, ele pode conter algum tipo de descrição dos dados que vêm a seguir.

Diferente dos dados escalares (contadores e temporizadores), os rastros aumentam de tamanho durante a execução, o que pode ser um problema. Além disso, quanto maior o nível de detalhamento desejado, maior a frequência com que dados são adicionados ao rastro. Isso, associado ao desejo de monitorar programas de longa duração, pode gerar dificuldades relacionadas ao espaço necessário para armazenar os rastros ou, no caso de um sistema *on-line*, a ocupação de banda de comunicação para transmití-los.

2.3.3 Vetores: Séries temporais

Um vetor temporal de dados de desempenho é um meio termo entre dados escalares e rastros. Um exemplo desse tipo de representação são os histogramas temporais da ferramenta Paradyn (MILLER et al., 1995). Um histograma temporal (*time histogram*) é um vetor contendo valores de uma medida de desempenho para uma sequência de intervalos de tempo. A granularidade dos dados é definida pelo tamanho do *bucket* (intervalo de tempo) e pelo número de *buckets*. Quando todos os *buckets* são ocupados, espaço para novas medições é liberado, através da duplicação do tamanho do *bucket* e condensação dos dados antigos. Dessa maneira, quanto mais tempo durar a execução do programa, mais grossa será a granularidade dos dados armazenados.

2.4 Monitoramento de Grid

Grid é um sistema distribuído que tem por objetivo fornecer compartilhamento e uso controlados e coordenados de recursos em organizações virtuais. Em tal ambiente, deve ser definido claramente e cuidadosamente o que é compartilhado, a quem é permitido o compartilhamento e as condições sob as quais esse ocorre. Um conjunto de indivíduos ou entidades definido por tais regras de compartilhamento é chamado organização virtual, comumente denominado pela sigla VO (de *virtual organization*, em inglês) (FOSTER;

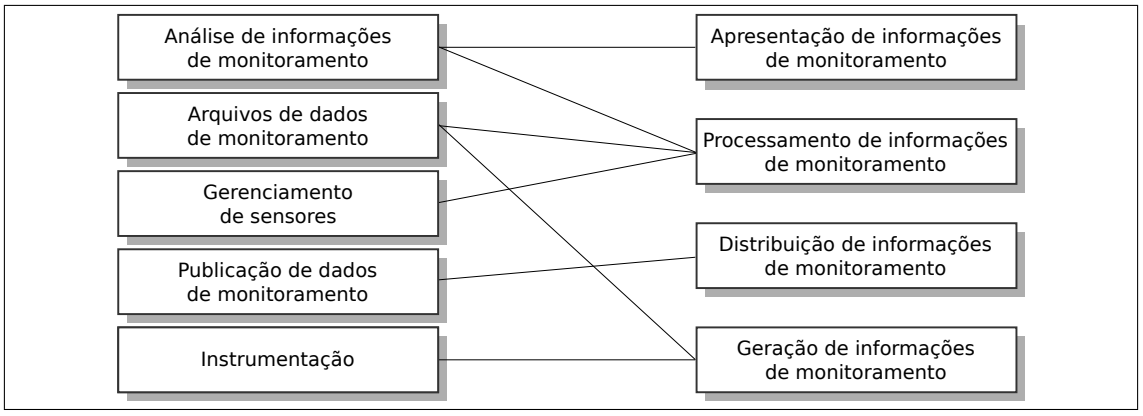


Figura 2.3: Relacionamento entre a visão do sistema de monitoramento por componentes e as funcionalidades citadas no capítulo 1.

KESSELMAN; TUECKE, 2001).

"Monitoramento de *Grid* é a medição e publicação do estado de um componente do *Grid* em um determinado instante" (HOLLINGSWORTH; TIERNEY, 2004). As funções necessárias para um sistema de monitoramento *Grid* são similares às abordadas anteriormente, para sistemas distribuídos em geral. Os dados precisam ser gerados, distribuídos, processados e apresentados. As informações contidas nas seções anteriores são válidas também para *Grid*. No entanto, há alguns aspectos específicos, que serão abordados nesta seção.

Uma maneira de representar um sistema de monitoramento de *Grid* é através de uma série de componentes que colaboram para sua realização. Esses componentes são responsáveis pela instrumentação, publicação de eventos de monitoramento, gerenciamento de sensores, manutenção de arquivos de eventos e análise das informações. A figura 2.3 mostra a relação entre esses componentes e os tipos de tarefas apresentados no capítulo anterior.

2.4.1 Requisitos para ferramentas de monitoramento de *Grid*

Muitos dos componentes citados na seção anterior são comuns em sistemas de monitoramento distribuídos tradicionais. No entanto, vários aspectos fazem com que o monitoramento seja mais difícil em ambientes de *Grid*. Um sistema de monitoramento de *Grid* possui os seguintes requisitos (HOLLINGSWORTH; TIERNEY, 2004):

- *Descoberta*: A distribuição geográfica de um sistema de *Grid* dificulta a descoberta dos componentes disponíveis e de como estão comportando-se. Há a necessidade de os consumidores de informações de monitoramento poderem encontrá-las e entendê-las. Para isso, pode-se utilizar um registro distribuído de produtores de dados de monitoramento através do qual os consumidores obtém informações sobre os dados disponíveis e informações de acesso ao mesmo. Então, os dados poderiam ser obtidos diretamente dos produtores ou através de um mecanismo de distribuição. Vários fatores podem influenciar a escolha técnica usada, como a arquitetura do sistema monitorado ou o fim para o qual as informações serão utilizadas.

- *Autorização (controle de acesso)*: É necessário haver um mecanismo de controle de acesso às informações de monitoramento. Por exemplo, uma organização pode não querer publicar externamente informações que possam comprometer a segu-

rança do sistema. Também pode-se querer limitar o acesso a algumas informações somente a usuários que comprovadamente necessitem-nas. Dessa forma diminui-se a possibilidade de sobrecarga do sistema devido à transmissão de grandes volumes de informações.

- *Interoperabilidade*: Seria interessante o uso de um formato comum de representação de eventos, de maneira a facilitar a interoperabilidade entre diferentes sistemas de monitoramento.

- *Intrusividade*: Como foi explicado no capítulo anterior, a intrusividade é um fator crítico para a instrumentação. A grande escala dos sistemas de *Grid* acentua ainda mais esse problema, pois há mais dados sendo gerados e transmitidos. Além disso, alguns recursos, como a rede de conexão, podem ser compartilhados entre vários usuários. Portanto, precisa-se evitar ao máximo a interferência entre atividades de monitoramento realizadas por diferentes usuários. Por esse mesmo motivo, também é maior a possibilidade de que o comportamento dos componentes observados seja modificado, de maneira indesejável, pelo monitoramento.

2.4.2 The Grid Monitoring Architecture

O *Open Grid Forum* (anteriormente chamado *Global Grid Forum*) elaborou uma especificação de uma arquitetura de monitoramento de *Grid*, chamada em inglês de *Grid Monitoring Architecture* (GMA) (TIERNEY et al., 2002). Essa arquitetura é uma tentativa de fornecer uma especificação mínima para suportar as funcionalidades necessárias a ferramentas de monitoramento de *Grid* e possibilitar a interoperabilidade entre elas.

Na figura 2.4 os componentes da GMA são mostrados: produtores, consumidores e serviço de diretório. Produtores publicam sua existência no diretório. Consumidores buscam no diretório informações sobre os produtores disponíveis. A GMA suporta três tipos de interação entre produtores e consumidores: *publish / subscribe*, *query / response* e notificação. O primeiro tipo consiste de três etapas: subscrição para o recebimento de um tipo de evento, criação de um fluxo de eventos do produtor para o consumidor e finalização da subscrição. O segundo tipo, *query / response*, consiste de interações iniciadas por um consumidor que realiza uma consulta, resultando em uma resposta do produtor. As interações do tipo notificação devem ser iniciadas por um produtor para notificar um evento a um consumidor. Na GMA os dados são transmitidos diretamente dos produtores para os consumidores. O serviço de diretório contém, junto com o registro dos produtores, informações necessárias para o estabelecimento da conexão.

Adicionalmente, um componente da GMA pode ser ao mesmo tempo consumidor e produtor. Nesse caso, tem-se uma espécie de republicador, através do qual um consumidor pode acessar informações provenientes de diversos produtores. Um outro uso dessa característica é a manutenção de um arquivo de informações de monitoramento. Ou seja, os dados recebidos dos produtores, podem ser armazenados persistentemente pelo componente. Dessa maneira, ter-se-ia um produtor capaz de fornecer dados históricos de monitoramento. Além disso, vários outros tipos de processamento, como alguns dos citados na seção 2.1.2, poderiam ser realizados a partir das informações agregadas pelo republicador. Um exemplo seria um republicador que forneça dados estatísticos relacionados às informações que consome de outros produtores.

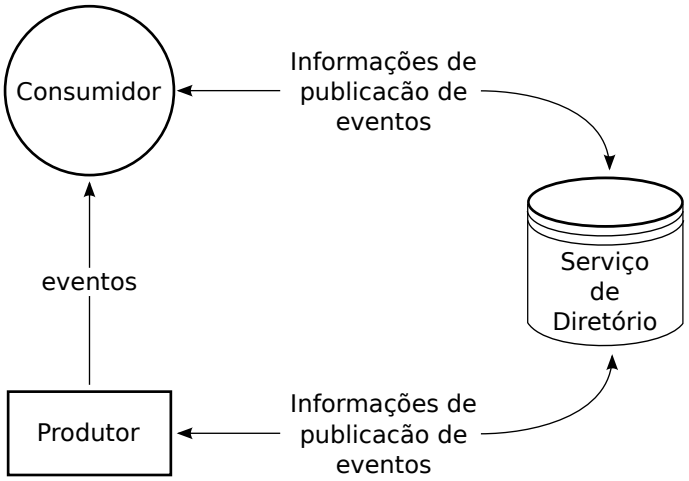


Figura 2.4: Componentes da GMA (Grid Monitoring Architecture)

2.4.3 A taxonomia de Zanikolas e Sakellariou

Zanikolas e Sakellariou publicaram uma taxonomia de sistemas de monitoramento de *Grid* (ZANIKOLAS; SAKELLARIOU, 2005). As categorias dessa taxonomia são representadas por níveis que variam de zero a três. Essas dependem da existência e características de publicadores e republicadores no sistema classificado. As características das ferramentas enquadradas em cada um dos quatro níveis, conforme ilustra a figura 2.5, são as seguintes:

- *Nível 0*: eventos trafegam diretamente dos sensores para os consumidores.
- *Nível 1*: eventos são acessíveis remotamente através de uma API fornecida por produtores. Nesse caso ou os sensores estão hospedados na mesma máquina que os produtores ou os produtores são sensores.
- *Nível 2*: além de produtores, há pelo menos um tipo de republicador, que tem uma funcionalidade fixa (filtragem, agregação ou arquivamento, por exemplo). Republicadores com diferentes funcionalidades podem ser "empilhados" uns sobre os outros, mas apenas de maneiras pré-definidas.
- *Nível 3*: além de produtores, há republicadores configuráveis, possibilitando que sejam organizados arbitrariamente em uma estrutura hierárquica.

A taxonomia também inclui um qualificador de multiplicidade usado para especificar se os republicadores dos sistemas de nível 2 são centralizados, simplesmente distribuídos ou distribuídos com suporte a replicação. Além disso, é levado em conta o tipo de entidade que é monitorada primariamente pelo sistema. Esse qualificador pode ser um dentre *hosts*, *redes*, *aplicações*, *disponibilidade* ou *genérico*. Outro qualificador empregado é o que diz se o sistema é *empilhável*. Sistemas com essa característica são aqueles projetados para utilizar produtores e republicadores de outros sistemas, operando em cima desses.

Para caracterizar os sistemas com base nos qualificadores explicados acima, utiliza-se uma nomenclatura no formato *L{0-3}.{H,N,A,V,G}.[S]*. O número representa o nível e a letra que vem a seguir representa o tipo principal de entidade monitorada: **H** para ***H****osts*; **N** para rede (***N****etwork*); **A** para **A**plicações; **V** para disponibilidade (*a**V**ailability*); **G** para sistemas **G**enéricos. No final, um **S** opcional identifica sistemas "empilháveis"

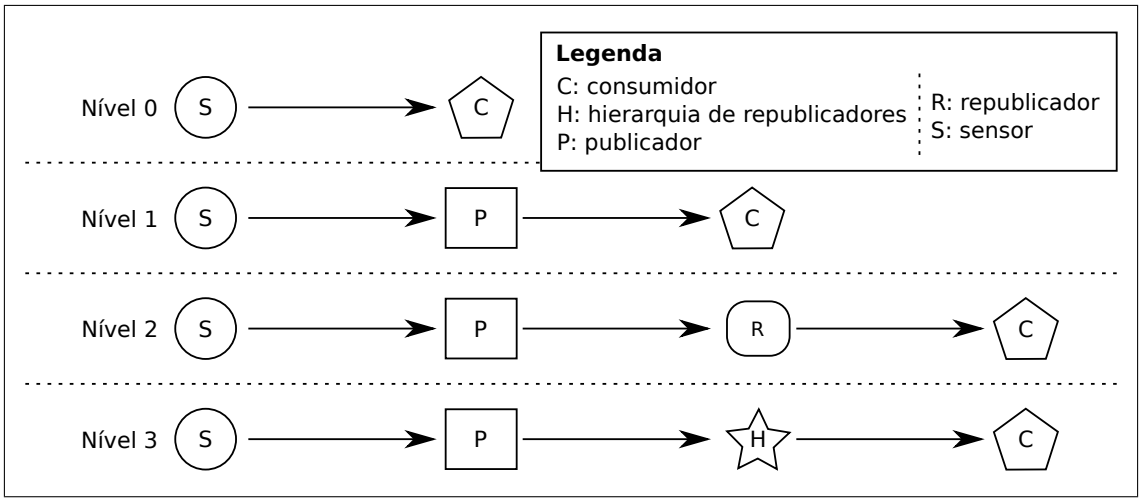


Figura 2.5: Níveis da Taxonomia de Zanikolas e Sakellariou.

(***S**tackable*). Além disso, apenas no caso dos sistemas de nível 2, uma letra **a**, **b** ou **c** logo após o número significa que o sistema tem, respectivamente: um republicador único e centralizado; mais que um republicador distribuído; republicadores distribuídos com suporte a replicação de dados. Dessa maneira, o código L2c.H caracteriza um sistema de nível 2 preocupado principalmente com o monitoramento de *hosts* e que possui mais de um republicador com suporte a replicação de dados.

2.5 Considerações Finais

Nesse capítulo foi abordado o monitoramento de sistemas distribuídos, que é a área em que está situado o trabalho aqui apresentado. Inicialmente foram abordadas as etapas envolvidas nessa tarefa: geração, processamento, distribuição e análise dos dados de monitoramento. Foram apresentadas funcionalidades que podem estar inclusas em cada uma dessas etapas. Também foram mostradas alternativas a serem consideradas em seu projeto.

Após, foram abordados dois aspectos a serem levados em conta na implementação de ferramentas de monitoramento de sistemas distribuídos. O primeiro deles é a intrusividade causada. Foi apresentada a relação dessa característica com três tipos de instrumentação: em hardware, em software e híbrida. Foram discutidas as vantagens e desvantagens de cada uma dessas. O segundo aspecto visto foram as dificuldades apresentadas na obtenção do estado global de um sistema distribuído; na sincronização do tempo entre os recursos; e na ordenação dos eventos. Por último, foram apresentadas formas de representação dos dados de monitoramento.

Também foram apresentados aspectos específicos do monitoramento de *Grid*. Além disso,foram apresentados alguns requisitos para ferramentas de monitoramento de *Grid*. A seguir, foi apresentada a *Grid Monitoring Architecture* (GMA), elaborada pelo *Open Grid Forum*. Essa é composta por produtores, consumidores e um serviço de diretório. Os produtores geram eventos e registram-se no serviço de diretórios. Os consumidores descobrem os produtores através desse serviço. Nesse modelo, os consumidores recebem notificações diretamente dos produtores.

Por último, foi vista uma taxonomia para classificação de sistemas de monitoramento de *Grid*. Essa leva em conta características como: a presença de uma API de acesso a

publicadores; a existência de republicadores; e a possibilidade de formar uma hierarquia arbitrária de republicadores.

No capítulo seguinte são abordados alguns trabalhos relacionados a esta dissertação. Inicialmente são apresentados o DIMVisual e o TRIVA, que são ferramentas utilizadas neste trabalho. Após, tem-se um estudo de um conjunto de sistemas de monitoramento.

3 FERRAMENTAS DE MONITORAMENTO

Nesse capítulo são apresentadas algumas ferramentas de monitoramento, iniciando com algumas utilizadas no presente trabalho. Primeiramente é descrito o DIMVisual. Esse é um modelo de integração de dados de monitoramento ao qual o modelo de coleta foi integrado. Após, tem-se uma visão geral do TRIVA, que é a ferramenta de visualização que é utilizada no protótipo implementado para avaliação do DIMVHCM. São apresentadas abordagens de visualização e outras técnicas implementadas pela ferramenta.

A seguir, na seção 3.3, é apresentado um estudo de alguns sistemas de monitoramento para sistemas distribuídos. São ressaltadas características dessas ferramentas, que servirão como base para uma comparação apresentada no capítulo 4 seção 4.6.

3.1 DIMVisual

O DIMVisual (SCHNORR; NAVAUX; OLIVEIRA STEIN, 2006) é um modelo de integração de dados de monitoramento para visualização. Nesse modelo, inicialmente os dados são coletados por diferentes ferramentas de monitoramento. Depois, esses dados passam por um processo de integração e são unificados, para poderem ser visualizados em conjunto, através de uma ferramenta de visualização.

O processo de integração, representado na figura 3.1, tem três etapas: sincronização, unificação e padronização. A sincronização é necessária devido aos dados provirem de diferentes origens. A segunda etapa consiste em unificar os identificadores utilizados pelas ferramentas de monitoramento conforme uma hierarquia de entidades (cluster, nó, processo, etc.). O último passo, consiste em converter a forma de registro de dados de cada ferramenta de coleta para um formato padrão que possibilite sua visualização.

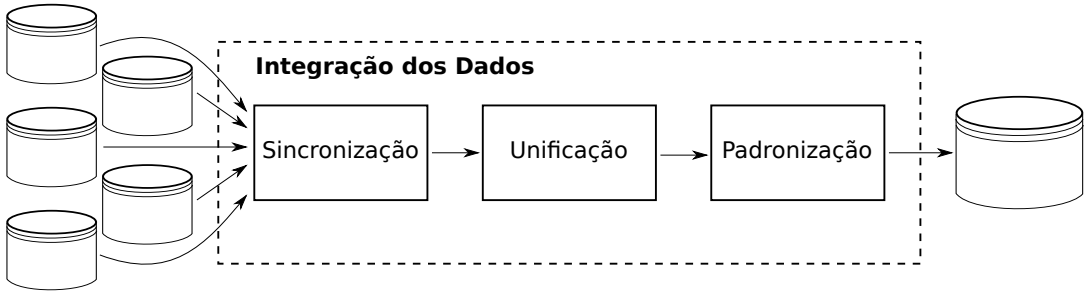


Figura 3.1: Modelo da integração de dados no DIMVisual

Existe um protótipo do DIMVisual que recebe arquivos de dados coletados por várias ferramentas e gera um arquivo, visualizável utilizando a ferramenta Pajé (KERGOMMEAUX; OLIVEIRA STEIN, 2000). Esse protótipo tem por componentes principais as

fontes de dados, o ordenador de dados, o controlador de integração e o controlador de aplicação, conforme mostra a figura 3.2.

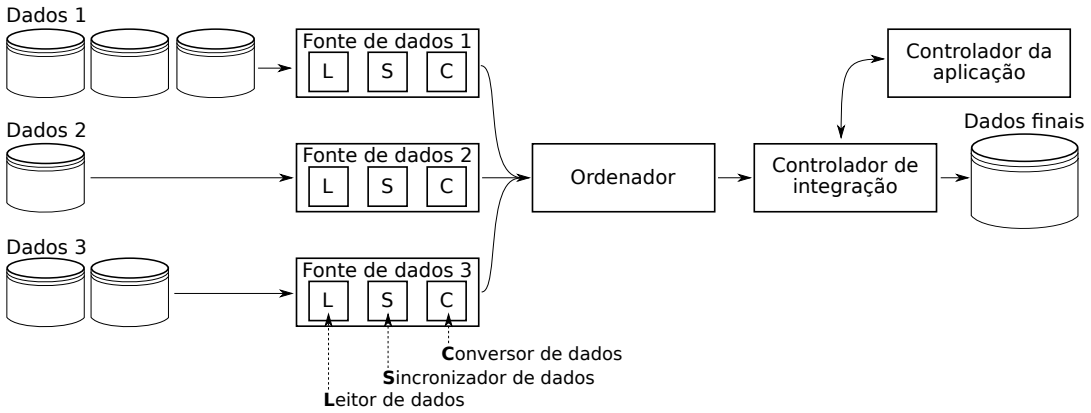


Figura 3.2: Arquitetura do protótipo do DIMVisual

Os componentes fonte de dados possuem um leitor, um sincronizador e um conversor. O primeiro é responsável pela leitura dos arquivos. O segundo corrige os tempos dos dados. O terceiro é responsável pela unificação dos identificadores e pela padronização dos dados. O ordenador gerencia as fontes de dados. Ele deve ler o evento com menor tempo dentre os disponíveis em todas as fontes. O controlador de integração faz a unificação hierárquica dos dados, fornece aos conversores os identificadores das entidades utilizadas, configura o sistema e grava em arquivo os resultados fornecidos pelo ordenador. Por fim, o controlador de aplicação fornece para o usuário uma interface para a configuração do protótipo.

O DIMVisual não especifica um mecanismo de coleta de dados em sistemas distribuídos. Isso significa que dados devem ser reunidos previamente à aplicação do modelo. O trabalho aqui implementado integra-se ao DIMVisual, tirando do usuário a responsabilidade de reunir os dados. Também é eliminada a necessidade de esperar pelo fim da execução para receber os dados.

3.2 TRIVA

TRIVA (*Three-dimensional Interactive Visualization Analysis*) (SCHNORR; HUARD; NAVAUX, 2008, 2009, 2010; SCHNORR; NAVAUX; HUARD, 2009) é um protótipo de ferramenta de visualização que implementa duas abordagens para visualização do comportamento de aplicações distribuídas. A primeira, utiliza representação gráfica em três dimensões para agregar informações sobre recursos à visualização do tipo espaço tempo. A outra, utiliza a técnica "Treemap" para representação hierárquica do comportamento de aplicações de grande escala.

A abordagem tridimensional possibilita a combinação da visualização espaço-tempo com informações topológicas e representação do padrão de comunicação da aplicação. Isso é feito através da utilização de dois eixos para a representação de recursos de hardware e da topologia da rede. O eixo que sobra (vertical) é utilizado para representar a evolução da entidade observada no tempo.

Dessa forma, às informações que seriam mostradas utilizando diagramas de Gantt são adicionadas informações sobre os recursos, na base da visualização. Essa abordagem tem a vantagem de possibilitar que o desenvolvedor analise facilmente as interações entre os

processos, assim como a relação da aplicação com a plataforma.

Dados de monitoramento podem conter informações sobre recursos de hardware (nós, processadores, etc.), assim como sobre componentes de aplicações (processos, threads, etc.). Essas entidades podem ser organizadas em uma hierarquia baseada em sua localização. Na abordagem que usa a técnica "Treemap", o TRIVA representa visualmente a estrutura hierárquica do sistema. A utilização dessa técnica permite mostrar o comportamento de aplicações paralelas de grande escala, devido ao bom aproveitamento do espaço disponível na tela.

Nessa abordagem, cada entidade é representada por um retângulo cujo tamanho relativo representa uma medida. Assim, os usuários podem comparar visualmente a relação entre os valores associados a cada entidade. Além disso, o usuário pode escolher para qual nível da hierarquia de entidades as informações serão mostradas. Os valores apresentados para um determinado nível são uma sumarização dos valores do nível logo abaixo. Por exemplo, pode-se visualizar informações sumarizadas por cluster. Nesse caso, os valores mostrados para um cluster são uma sumarização dos referentes aos nós que o compõe. Devido a esse mecanismo, pode-se dizer que o TRIVA suporta a visualização das informações em diferentes níveis de abstração.

Outro mecanismo presente no TRIVA é o algoritmo de integração por *time-slice*. Esse possibilita a visualização da integral dos valores contidos em uma fatia de tempo. Essa fatia pode ser ajustada pelo usuário, através da especificação de seu tamanho e data de início. Através dessa técnica, obtem-se uma sumarização do comportamento das entidades dentro do período selecionado. Além disso, o *time-slice* pode ser movido para observar a evolução das medidas no tempo.

Além disso, o TRIVA possibilita ao usuário a seleção das informações a serem visualizadas. Isso é feito através de uma interface gráfica. Nessa, o usuário pode escolher o conjunto de entidades a serem mostradas. Exemplos de entidade seriam clusters, máquinas, ou até tipos de eventos e medidas.

A figura 3.3 mostra a estrutura de componentes do TRIVA. Essa é composta por um controlador (`TRIVAController`) e uma interface gráfica. O controlador é reponsável pela inicialização de um pipeline de componentes, também mostrado na figura. Os dados são lidos de arquivos pelo primeiro componente (`FileReader`). Então, eles são repassados aravés dos componentes do pipeline. Cada um desses realiza parte do processamento necessário para gerar a visualização. Além disso, eles podem adicionar funcionalidades à ferramenta. Por exemplo, o componente `TypeFilter`, adiciona a funcionalidade de seleção de informações, mencionada no parágrafo anterior. O TRIVA possui diferentes configurações do pipeline de acordo com o tipo de visualização gerada.

Na próxima seção é estudado um conjunto de ferramentas de monitoramento. Essas ferramentas de monitoramento diferenciam-se do DIMVisual e do TRIVA por integrarem várias etapas do monitoramento, como geração, distribuição, processamento e apresentação das informações. Assim, elas podem ser consideradas sistemas de monitoramento. O protótipo apresentado no capítulo 5, integra DIMVisual e TRIVA ao DIMVHCM, para formar um sistema de monitoramento. No entanto cada um desses realiza apenas uma parte do processo.

3.3 Estudo de Sistemas de Monitoramento

Nessa seção são apresentadas ferramentas de monitoramento que formam sistemas de monitoramento completos. Ou seja realizam todas as etapas envolvidas nesse processo.

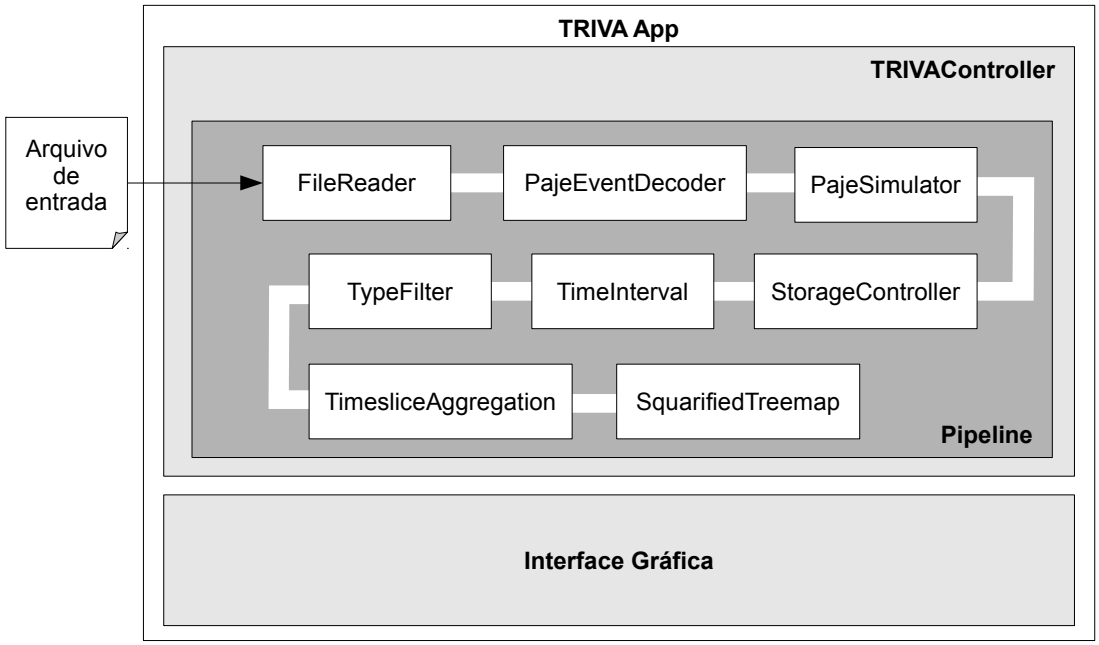


Figura 3.3: Diagrama dos componentes do TRIVA

A maioria dessas são ferramentas de monitoramento de *Grid*, cujo estudo é interessante devido aos desafios apresentados por essa plataforma.

Existem várias ferramentas de monitoramento que são utilizadas em *Grid*. Algumas delas foram desenvolvidas com outro objetivo ou para outros tipos de sistema, mas acabaram sendo usadas também em *Grid*. Entre essas estão Ganglia (MASSIE; CHUN; CULLER, 2004; SACERDOTI et al., 2003), que foi desenvolvida inicialmente para o monitoramento de *clusters* e *Network Weather Service* (NWS) (WOLSKI; SPRING; HAYES, 1999) que é um serviço de previsão de desempenho para redes de computadores. Outras ferramentas são específicas para Grid, como GridICE (ANDREOZZI et al., 2005; AIFTI-MIEI et al., 2007) e o Monitoring and Discovery Service (MDS) (CZAJKOWSKI et al., 2001), que faz parte do Globus Toolkit (The Globus Alliance, 2008; FOSTER, 2005a). Os desafios enfrentados no monitoramento desses sistemas, tornam interessante o estudo dessas ferramentas. Outra ferramenta interessante é a Monalytics (KUTARE et al., 2010), que foi projetada para auxiliar no gerenciamento de *data centers* de grande escala. Uma de suas principais característica é a realização do processamento e análise perto da origem dos dados.

Nessa seção são apresentadas algumas dessas ferramentas. As principais características levadas em conta na análise realizada são apresentadas na subseção a seguir. Logo após, tem-se seções dedicadas às ferramentas. A subseção 3.3.2 trata da ferramenta Ganglia, a 3.3.3 da GridICE, a 3.3.4 da MonALISA e a subseção 3.3.5 refere-se ao Globus MDS. Por último, tem-se o estudo da ferramenta Monalytics, na seção 3.3.6.

3.3.1 Características Analisadas

No estudo das ferramentas de monitoramento, apresentadas nas próximas subseções, serão levadas em conta as seguintes características:

- *Objetivo principal*: as escolhas feitas durante o projeto de uma ferramenta de monitoramento devem levar em conta as tarefas para as quais pretende-se utilizá-la.

Por exemplo, um objetivo pode ser a geração de alertas quando uma condição for satisfeita, como a ocorrência de falhas. Nesse caso, pode ser interessante que se tenha um mecanismo baseado em eventos para que o alerta seja emitido assim que a condição for satisfeita. Já no caso da análise *post-mortem* da execução de aplicações, tal mecanismo pode não ser tão necessário. Para essa tarefa, o fornecimento de informações sob demanda pode ser utilizado.

- *Estrutura de distribuição de informações de monitoramento*: a maneira como as informações de monitoramento são distribuídas para seus usuários é uma característica importante de um sistema de monitoramento distribuído. Quanto ao tempo, os dados podem ser transmitidos assim que se tornarem disponíveis, ou que uma determinada quantidade de informação houver sido coletada localmente ou então apenas quando os consumidores solicitarem-nas (sob demanda). Nos dois primeiros casos utiliza-se um mecanismo de recebimento de dados do tipo *push*, pois o produtor dos dados inicia sua transmissão. No terceiro caso é necessária a ação do consumidor para que a transmissão seja iniciada (*pull*). No caso de mecanismos do tipo *push*, pode ser utilizado um esquema *publish / subscribe* para aumentar o dinamismo do sistema e também possibilitar a filtragem implícita dos dados, conforme foi dito na seção 2.1.2. Além disso, os dados podem ser transmitidos diretamente dos produtores para seus consumidores ou através de republicadores intermediários. No caso do uso de republicadores, características interessantes são a maneira como esses são estruturados e suas funções. Outro aspecto a ser levado em conta nesse tópico é a maneira como é feita a descoberta dos recursos e das informações de monitoramento disponíveis.

- *Manutenção de arquivos de informações de monitoramento*: a manutenção de arquivos de monitoramento é interessante para algumas tarefas como *profiling*, análise de desempenho e de falhas. Alguns aspectos que podem ser levados em conta quanto a essa funcionalidade são o tempo de vida das informações, o nível de detalhamento e a utilização de sumarização para a redução do tamanho de dados antigos. Essas características devem ser adequadas aos objetivos da ferramenta. Um aspecto extremamente importante é o espaço necessário para armazenar o grande volume de informações de monitoramento que pode ser gerado por um sistema formado por milhares de recursos. Nesse caso, o uso de arquivos distribuídos pode ser cogitado.

- *Possibilidade de integração com outras ferramentas*: alguns sistemas de monitoramento são capazes de receber dados provenientes de outras ferramentas. Para isso, na maioria das vezes esses dados devem estar em um formato padrão e autodescritivo que pode ser específico ou não ao sistema que os recebe. Em alguns casos é necessária a implementação de um conversor para que as informações sejam entendidas.

- *Apresentação das informações*: as informações de monitoramento podem ser apresentadas graficamente ou através de API's que possibilitem seu uso por outras aplicações. A apresentação gráfica das informações pode possibilitar a visualização das informações em múltiplos níveis de abstração. Além disso, pode ser possível avançar e voltar a visualização com relação ao tempo. Também pode haver a possibilidade de escolha dos recursos sobre os quais se quer obter informações. Outro tipo de informação que pode ser mostrada são as interações entre os objetos monitorados.

3.3.2 Ganglia

Ganglia (MASSIE; CHUN; CULLER, 2004) é uma ferramenta de monitoramento distribuído voltada para sistemas de computação de alto desempenho, como *clusters* de computadores e *Grids*. Ela é baseada em uma estrutura hierárquica voltada a *federações de clusters*. Internamente a um *cluster*, ela utiliza um protocolo baseado em multicast. Externamente, os dados originados em diferentes *clusters* são integrados usando uma árvore de conexões ponto a ponto.

A arquitetura da ferramenta é mostrada na figura 3.4. Cada nó possui um serviço chamado *Ganglia monitoring daemon* (*gmond*). Esse serviço coleta as informações locais e as transmite para todos os nós do *cluster*, utilizando um protocolo *multicast*. Dessa forma, qualquer nó de um *cluster* possui as informações de monitoramento de todos os nós que o compõe. Portanto, internamente ao cluster utiliza-se um modelo *push* de publicação de dados de monitoramento. Além disso, dentro de um *cluster*, o controle de quais nós estão ativos é feito através de um *heartbeat* enviado pelos nós através do canal de *multicast*.

Além dos *gmond*, o Ganglia possui serviços chamados *Ganglia Meta Daemon* (gmetad), que formam uma árvore de conexões ponto-a-ponto para agregar dados de múltiplos *clusters*. Cada *gmetad* da árvore busca periodicamente dados de um filho que pode ser outro *gmetad* ou um *gmond* executando em um nó de um *cluster*. Dessa maneira, os *gmetad* utilizam um modelo *pull* para a publicação de dados.

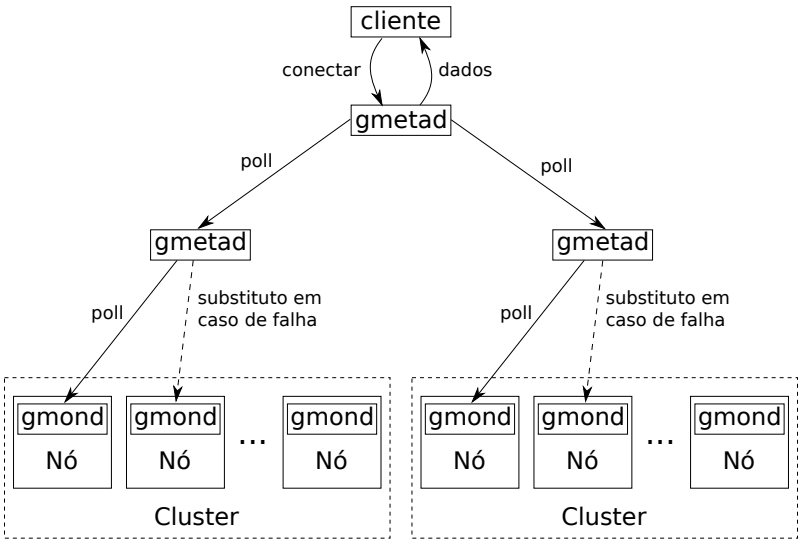


Figura 3.4: Arquitetura da ferramenta de monitoramento Ganglia, mostrando os serviços *gmond* e *gmetad*

O Ganglia utiliza RRDtool (*Round Robin Database*) (Tobias Oetiker, 2008) para armazenar dados históricos de monitoramento. Com essa ferramenta é possível armazenar dados em arquivos de tamanho limitado, com diferentes granularidades. Dados mais recentes podem ser armazenados em um arquivo com maior granularidade. Enquanto mais antigos podem ser sumarizados e armazenados em um arquivo com menor granularidade. Dessa forma tem-se um maior controle sobre a quantidade de dados armazenados.

RRDtool também é responsável pela visualização dos dados que armazena. Podem ser gerados gráficos dos dados em diferentes granularidades com relação ao tempo. Esses gráficos são apresentados aos usuários através de um *front-end* web. A interatividade da visualização é bastante limitada. Não se pode avançar e voltar a visualização com relação ao tempo. Devido à maneira como o RRDTool funciona, é impossível selecionar um

determinado período de tempo a ser visualizado. Apenas é possível selecionar períodos como ultima hora, último mês, última semana. Também não se pode selecionar um grupo arbitrário de componentes a serem visualizados. Além disso, não há suporte para mostrar interações entre os objetos observados, devido à RRDtool apenas plotar séries históricas.

Ganglia foi aperfeiçoado para melhorar sua escalabilidade em sistemas geograficamente distribuídos e seu desempenho (SACERDOTI et al., 2003). Dentre as mudanças, cada *gmetad* passou a armazenar dados detalhados apenas sobre *clusters* locais a ele. *Clusters* locais são aqueles que estão ligados diretamente ao *gmetad*. Apenas informações sumarizadas com relação aos outros *clusters* são guardadas pelos *gmetad*. Além disso, foi adicionado um motor de consultas ao sistema. Dessa maneira pode-se obter informações sobre uma subárvore específica. Devido ao custo em desempenho, a frequência com que os dados são buscados para serem sumarizados é muito baixa (a cada 15 segundos). Dependendo da utilização pretendida das informações de monitoramento, isso pode ser um problema.

3.3.3 GridICE

GridICE (ANDREOZZI et al., 2005; AIFTIMIEI et al., 2007) é uma ferramenta projetada com o objetivo de preencher uma série de requisitos baseados em um conjunto de casos de uso. Esses casos focam-se nas preocupações de uma VO, de um administrador local de um *site* e de uma organização encarregada de coordenar o funcionamento do *Grid*. O termo *Grid Operations Center* (GOC) é utilizado para denominar essa organização. Os requisitos levantados são os seguintes:

- particionar dinamicamente recursos e uso de serviços usando como critérios a propriedade do *site*, o domínio de operação e a acessibilidade por organizações virtuais;
- coletar dados em ordem para possibilitar análise retrospectiva;
- lidar com um grande volume de dados utilizando mecanismos de redução;
- coletar dados com diferentes granularidades;
- ajudar na detecção da ocorrência falhas e possivelmente preveni-las;
- fornecer funcionalidades gerais de visualização e análise;
- basear-se em um modelo comum de informação sobre recursos de *Grid*;
- adotar interfaces e protocolos padrão dentro da comunidade de *Grid*;
- integrar-se com sistemas locais de monitoramento, quando disponíveis;
- acompanhar quais máquinas estão executando as aplicações de uma VO, o estado e comportamento de cada máquina e o comportamento do software.

A arquitetura dessa ferramenta é estruturada em cinco camadas, conforme mostra a figura 3.5. A camada inferior é o serviço de medição, que obtém medidas dos recursos monitorados. Os dados obtidos são armazenados localmente em um repositório do *site*. Isso evita perda dos dados no caso de uma partição da rede. Os dados são armazenados utilizando um modelo comum de informação, para que dados de diferentes *sites* possam ser correlacionados e integrados.

Figura 3.5: Camadas da arquitetura da ferramenta de monitoramento GridICE

A segunda camada é o serviço de publicação, cujo papel é fornecer dados aos consumidores. Devido a nem todos os nós terem acesso à Internet, os dados são coletados localmente em um nó que é capaz de fazê-lo. O *Grid Information Service* (GIS), que faz parte do *Globus Monitoring and Discovery Service* (MDS) versão 2, é utilizado como interface de acesso aos dados de monitoramento. O MDS é abordado com mais detalhes na seção 3.3.5. Apenas dados necessários para a descoberta dos dados de monitoramento são propagados na hierarquia do MDS. As informações de monitoramento são acessadas diretamente no *site* pelos consumidores. O objetivo dessa estratégia é diminuir o carregamento da raiz da hierarquia. Um detalhe importante é que essa hierarquia utiliza um modelo *pull*, portanto é necessário *polling* contínuo para a descoberta da disponibilidade de novos dados de monitoramento.

A camada seguinte corresponde ao serviço de coleta de dados, que possibilita a coleta de dados históricos e é formada por vários componentes. O componente de detecção de novos recursos, o de escalonamento e o de armazenamento persistente são os principais. O primeiro observa periodicamente o GIS para detectar novas fontes de dados de monitoramento. O de escalonamento inicia observações periódicas para obter os dados de monitoramento oferecidos e guarda os resultados utilizando o componente de armazenamento persistente. Esse último foi implementado utilizando o sistema de gerenciamento de banco de dados PostgreSQL.

A quarta camada possui dois serviços: o serviço de detecção e notificação e o de análise de dados. O primeiro é responsável por prover um meio de descrever eventos, por detectá-los e por enviar notificações de sua ocorrência aos usuários adequados. O segundo componente fornece análise de desempenho, nível de uso e estatísticas e relatórios gerais.

A última camada trata-se do serviço de apresentação, que provê informação de monitoramento concisa através de uma interface gráfica via web. Essa interface possui diferentes visões, dependendo do tipo de consumidor. Algumas visões são: a visão de GOC, que compreende todos os recursos de *Grid* gerenciados pelo mesmo; a visão de *site*, que compreende os recursos pertencentes a um *site* específico; a visão de organização virtual, que compreende os recursos que podem ser acessados por usuários de uma determinada organização virtual. A interface principal mostra os últimos dados obtidos. Também há a possibilidade de visualizar gráficos comparando valores de medidas em diferentes *sites* tendo como base as médias em um determinado período de tempo. Além disso pode-se ver séries temporais relacionadas ao número de *jobs*. Não é possível navegar na visualização com relação ao tempo.

3.3.4 MonALISA

MonALISA (Monitoring Agents in a Large Scale Integated Service Architecture) (NEWMAN et al., 2003; LEGRAND et al., 2004) é uma ferramenta inicialmente desenvolvida para ser utilizada em redes e sistemas de *Grid* que suportam o processamento e análise de dados relacionados a experimentos globais colaborativos na área das físicas de altas energias e nuclear. O foco principal da ferramenta está ligado ao gerenciamento de *Data Grids* globais. Além disso, segundo os autores da ferramenta, as informações de monitoramento coletadas são essenciais para o desenvolvimento de serviços de *Grid* de mais alto nível e de componentes que forneçam apoio a decisões ou até algum nível de tomada automática de decisões.

Essa ferramenta é baseada em uma Arquitetura de Serviços Distribuídos Dinâmicos (NEWMAN; LEGR; BUNN, 2001), cuja sigla em inglês é DDSA (*Dynamic Distributed Services Architecture*). Ela possui um conjunto de *Station Servers*, sendo um por *site* de um *Grid*, que hospedam serviços dinâmicos baseados em agentes. Os serviços são descobertos mutuamente através de um serviço de busca. Eventos sinalizando mudanças de estado são notificados automaticamente aos serviços. A informação monitorada pode ser reportada a serviços de nível mais alto, que analisam as informações e realizam ações corretivas, visando melhorar a eficiência do sistema ou mitigar problemas.

Um serviço na DDSA é um componente que interage autonomicamente com outros serviços através de *proxies* dinâmicos ou via agentes que usam protocolos autodescritivos. Eles também podem utilizar-se de subscrição dinâmica a eventos, através da qual é possível registrar-se para ser notificado da ocorrência de um conjunto de tipos de eventos. Um protótipo da DDSA foi implementado, tendo como base a tecnologia JINI. Também foram incluídos bindings WSDL/SOAP para permitir o acesso às informações monitoradas por outros clientes.

MonALISA foi projetada para integrar facilmente outras ferramentas de monitoramento e fornecer a informação de maneira dinâmica e autodescritiva para qualquer serviço ou cliente. Módulos são usados para coletar diferentes informações ou para interagir com outras ferramentas de monitoramento. Esses são carregados dinamicamente e executados em diferentes *threads*. Um *pool* de *threads* é utilizado com o objetivo de reduzir a carga no sistema monitorado.

Cada serviço registra-se em um conjunto de *JINI Lookup Discovery Services* (LUS). Um LUS, por sua vez, pode ser registrado em outro LUS. No ato de seu registro é fornecido o código para os proxies que devem ser instanciados para acessar o serviço. O registro é baseado em um mecanismo de *lease* que deve ser renovado periodicamente. Caso um serviço falhe em fazê-lo, ele é removido do LUS e os serviços inscritos para receber eventos do mesmo são notificados. O mecanismo de descoberta é usado para anunciar quando novos serviços tornam-se disponíveis e quando esses ficam indisponíveis. Para receber informações de monitoramento, o cliente conecta-se diretamente aos serviços que as fornecem. Para isso, ele utiliza os *proxies* registrados no serviço de descoberta.

MonALISA possui uma interface gráfica global, mostrada na figura 3.6, implementada na forma de uma aplicação *Web Start*. Ela pode ser iniciada facilmente a partir de um navegador web. A tela inicial apresenta um mapa mundi mostrando a localização dos *sites* monitorados e suas interconexões. Além disso, ela possibilita visualizar parâmetros globais para múltiplos *sites*, assim como acompanhar detalhadamente parâmetros de qualquer *site* individual ou componente do sistema. Gráficos históricos e em tempo real podem ser gerados.

Os dados de monitoramento são armazenados em um banco de dados relacional local

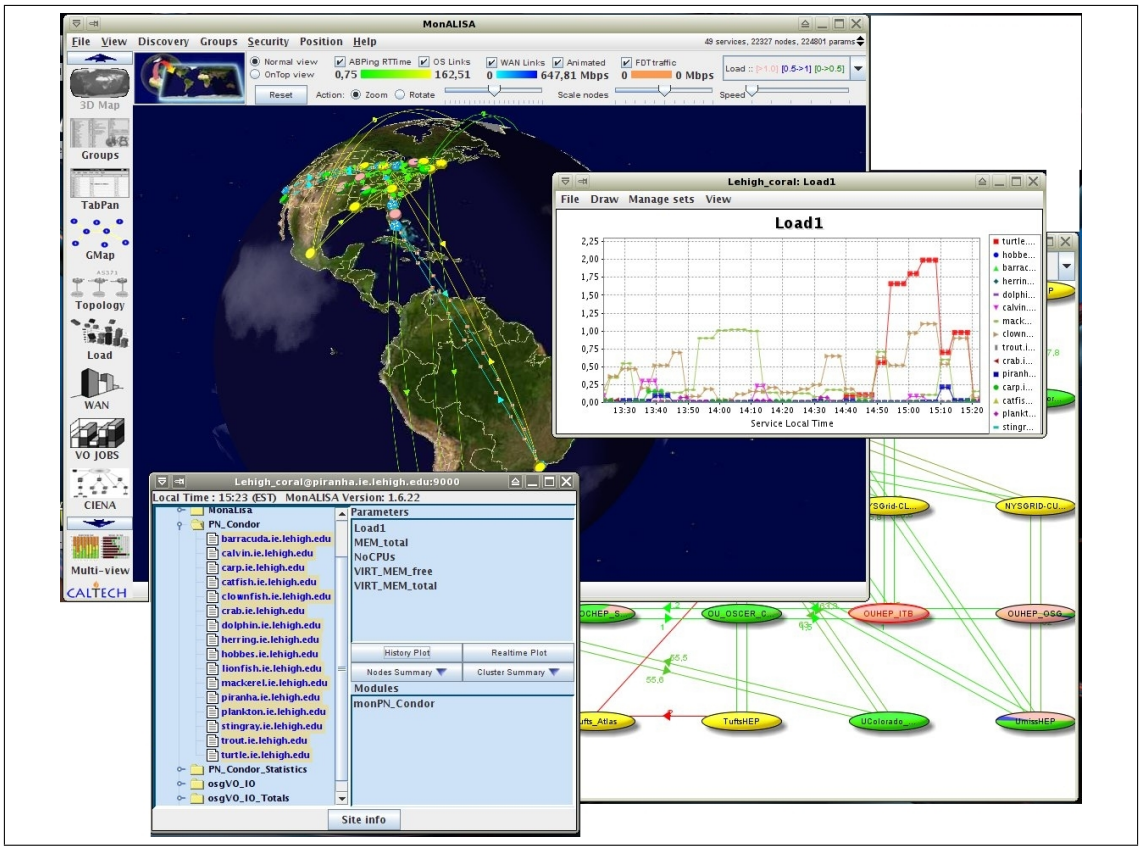


Figura 3.6: Interface gráfica *Web Start* da ferramenta MonALISA

a cada serviço. Dados mais antigos são comprimidos através do cálculo de suas médias e armazenados em conjunto com o intervalo de flutuação dos valores. Também foi desenvolvido um *framework* para a construção de pseudoclientes que utilizam os LUS's para encontrar os serviços, inscrevem-se pra receber dados do mesmo e os armazenam em um banco de dados local. A partir desse, o cliente pode fazer uso das informações, da forma que precisar.

3.3.5 Globus Monitoring And Discovery Service

O *Globus Toolkit* (The Globus Alliance, 2008) é um conjunto de bibliotecas e programas que tratam de problemas comuns relacionados à construção de serviços e aplicações para sistemas distribuídos. Ele possui um componente chamado *Monitoring and Discovery Service* (MDS) que tem por objetivo tratar, como o nome em inglês diz, dos problemas de descoberta e monitoramento.

Czajkowski et al. (CZAJKOWSKI et al., 2001) citam uma série de exemplos de uso de serviços de informações que suportem a descoberta e monitoração de recursos, serviços, computações e outras entidades que fazem parte de um *Grid*. Esses exemplos são um serviço de descoberta de serviços, um escalonador, um serviço de seleção de réplicas, um agente de adaptação de aplicações, um serviço de solução de erros e uma ferramenta de diagnóstico de desempenho. Os autores concluem que em todos esses casos tem-se uma estrutura similar: um ou mais consumidores desejam obter informações de um ou mais produtores. Então, eles propõem que esses casos podem ser tratados com um *framework* único e consistente. Isso teria vantagens significativas devido à necessidade inevitável de combinar aspectos dos cenários apresentados e de outros, de várias maneiras diferentes.

A implementação de um serviço desses em *Grid* deve levar em conta os desafios impostos pela diversidade dos recursos envolvidos, a abrangência das consultas aos dados necessárias e a dinamismo da participação em VO's e do estado dos recursos.

O serviço proposto é composto por dois elementos básicos. Um deles é um grande conjunto de *information providers* distribuídos que fornecem acesso a informações sobre entidades individuais. Esses utilizam um modelo padrão para formatar as informações. O outro são serviços de nível superior, responsáveis por coletar, gerenciar, indexar e/ou reagir a informações fornecidas por um ou mais *information providers*.

Interações entre serviços superiores e *providers* utilizam dois protocolos básicos: de registro e de *enquiry*. O primeiro é usado pelos *providers* para notificar a sua existência. O outro é utilizado para obtenção de informações através de consultas (sob demanda) ou subscrição. O serviço deve integrar-se ao *Grid Security Infrastructure*, que é um componente do *Globus Toolkit* que trata de aspectos relacionados à segurança, para realização de autenticação e controle de acesso.

Foi realizada uma implementação desse serviço de informações como parte do Globus Toolkit, chamada MDS-2. Dois tipos de serviços foram implementados: um *information provider*, chamado *Grid Resource Information Service* (GRIS); um *aggregate directory* reconfigurável, chamado *Grid Index Information Service* (GIIS). O *information provider* utiliza dois protocolos básicos. O primeiro é o *Grid Information Protocol* (GRIP), utilizado para acessar informações sobre entidades. O segundo é o *Grid Registration Protocol* (GRRP), utilizado para notificar os *aggregate directory services* sobre a disponibilidade de suas informações. Um *aggregate directory* é um serviço que usa GRIP e GRRP para obter informações sobre entidades e responder a consultas relacionadas a elas. Esses serviços fornecem um ponto único de contato para o qual consultas sobre um conjunto de objetos observados são destinadas. Dessa forma, cada *aggregate directory* fornece um diferente escopo de busca.

O protocolo GRIP suporta descoberta e consulta a informações. O MDS-2 utiliza LDAP para implementá-lo. São suportadas consultas especificando uma busca, localização ou subscrição, e respostas contendo campos especificados ou qualquer objeto. O protocolo GRRP define um mecanismo de notificação. Através desse, um serviço pode fazer *push* de informações sobre sua existência para outro elemento da arquitetura. Um *information provider* define em quais *aggregate directories* vai registrar-se, de acordo com as políticas locais de VO. O *information provider* mantém um fluxo de mensagens para cada diretório. Depois de um determinado tempo sem receber mensagens de um *information provider*, um diretório assume que esse está indisponível.

Aggregate Directories podem ser criados utilizando-se os protocolos GRRP e GRIP. Dessa maneira pode-se ter um serviço hierárquico de descoberta, com uma estrutura similar à mostrada na figura 3.7. Nesse caso, um diretório funciona como um *information provider* contendo informações sobre os recursos abaixo dele na hierarquia. Para construir a hierarquia, os diretórios usam GRRP para registrar-se com diretórios de nível superior ao seu. Também pode-se construir serviços especializados em que os diretórios utilizam GRIP para obter informações mais detalhadas sobre os recursos. Nesse caso, o diretório pode utilizar esquemas de armazenamento e atualização otimizados para usos específicos.

Serviços de descoberta de recursos geralmente necessitam apenas da informação sobre quais recursos de um conjunto estão disponíveis em um determinado instante. Por isso o uso de um modelo *pull* (sob demanda) para o recebimento de informações é mais adequado. Isso muda quando o serviço é destinado a realização de tarefas de monitoramento. Nesse caso frequentemente deseja-se observar a variação de determinadas caracterísi-

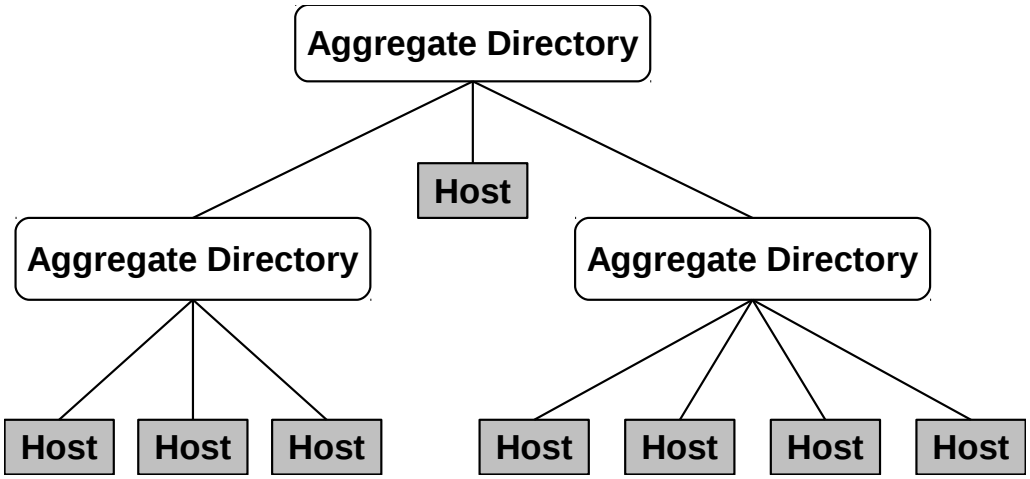


Figura 3.7: Exemplo de estrutura hierárquica formada por *Aggregate Directories*

cas de um conjunto de entidades ao longo de um período de tempo. Para essas tarefas, um modelo *push* que forneça o recebimento assíncrono de informações de acordo com condições pré-especificadas é mais adequado (CZAJKOWSKI et al., 2001). O protocolo GRIP prevê o uso de ambos modelos de recebimento de informações. Dessa maneira, ele suporta tanto descoberta como monitoramento.

Quanto à segurança, assume-se que um *information provider* pode especificar, para cada ítem de informação, as credenciais que devem ser apresentadas para acessá-lo. Para isso é usado um componente do *Globus toolkit* chamado *Grid Security Infrastructure* para verificar credenciais e obter autenticação mútua entre consumidores e fornecedores (*information providers*).

A implementação do MDS-2 não suporta o modelo *push* de recebimento de informações. Por esse motivo seu uso para monitoramento fica prejudicado. Para realizar as ações relativas ao protocolo GRRP é utilizado LDAP. Para tratar o aspecto da segurança, foi feita a integração do GSI com o OpenLDAP. Foi implementado um *framework* configurável para *information providers* chamado *Grid Resource Information Service* (GRIS). Além disso, resultados provenientes de um *provider* são mantidos em um *cache* por um período de tempo configurável, com o objetivo de reduzir o número de chamadas aos *providers*. Testes mostram que esse *cache* proporciona bons resultados relacionados à escalabilidade do MDS-2 (ZHANG; FRESCHL; SCHOPF, 2007). Outro *framework*, chamado *Grid Index Information Service* (GIIS), foi implementado para a construção de *aggregate directories*. Esse *framework* aceita mensagens GRRP enviadas por GRIS e GIIS filhos. Clientes podem fazer buscas em um GIIS para obter informações de qualquer ou de todos os filhos de um GIIS. Tanto GRIS quanto GIIS foram implementados na forma de *backends* do LDAP.

O MDS versão 4 (MDS-4) (FOSTER, 2005b) é a implementação do serviço de informações do Globus Toolkit 4 (GT4) (FOSTER, 2005a). Ele possui por componentes principais os *aggregator services* e os *information sources*. Também são fornecidas interfaces baseadas em navegador web, ferramentas de linha de comando e interfaces de Web Services para que os usuários possam consultar e acessar as informações coletadas.

Um *information source* é qualquer entidade da qual um *aggregator service* pode obter informação (arquivo, programa, Web Service, etc.). Os *information sources* necessitam registrar-se em um agregador para poderem ser descobertos e acessados. Esses registros têm um tempo de vida, ou seja, eles expiram caso não sejam renovados periodicamente.

Esse procedimento é feito via Web Services, de dois modos. No primeiro, o *information source* é registrado através do fornecimento, pelo usuário, de um programa a ser executado periodicamente para obter informação atualizada. O programa deve converter dados não XML para uma representação XML apropriada. O segundo modo é destinado a Web Services compatíveis com WSRF. Nesse caso, o usuário informa no registro se o agregador deve usar *pull* ou subscrição a mudanças nas propriedades do recurso. Os *aggregator services* coletam informações de estado recentes de todos os *information sources* registrados. Toda a informação obtida é tornada disponível através de uma interface Web Services. Há três tipos diferentes de *aggregator services*:

- *MDS-Index*: Torna disponíveis, como documentos XML, os dados coletados dos *information sources*. Podem ser realizadas consultas XPath nos últimos valores obtidos. Usuários podem escrever aplicações para obter informações desses agregadores, usando interfaces Web Service para as quais o GT4 fornece API's em C, Java e Python. Além disso, a ferramenta de linha de comando *wsrf-get-property* pode ser usada para obter informações de recursos. Há também uma interface chamada WebMDS que apresenta as informações de um MDS-Index em um navegador Web padrão. Essa é configurável usando transformações XSLT, sendo que o GT4 fornece algumas transformações padrão.

- *MDS-Trigger*: Realiza ações especificadas pelo usuário, quando informações coletadas satisfazem critérios especificados pelo mesmo. Isso é feito através de uma interface Web Service que permite ao usuário definir uma consulta XPath e um programa. Esse será executado sempre que que um novo valor adequar-se à regra fornecida pelo usuário.

- *MDS-Archiver*: Armazena valores dos *information sources* em um banco de dados persistente, que pode ser consultado por clientes para a obtenção de informações históricas. Clientes podem especificar um intervalo de tempo para o qual são necessários valores.

O MDS-4 faz uso pesado de XML e interfaces Web Service. Toda a informação coletada por um agregador é mantida em formato XML. Essa pode ser consultada usando XPath ou outros mecanismos de Web Services. Em relação ao MDS-2, tem-se algumas diferenças importantes (FOSTER, 2005b): uso da linguagem de consulta XPath ao invés de LDAP; implementação mais simples e robusta devido a haver menos componentes; configuração simplificada, devido a menos componentes e boa integração com o GT4; interface conveniente para o uso de *information sources* arbitrários, devido à extensibilidade da arquitetura; não há requisitos quanto ao uso esquemas pré definidos pelos information providers; o desempenho é menor, devido ao uso de XML.

3.3.6 Monalytics

Monalytics (KUTARE et al., 2010) refere-se a um sistema que combina monitoramento e análise. Seu objetivo é auxiliar no gerenciamento de *data centers* de grande escala. Uma de suas principais características é a realização da análise "perto" das fontes de dados (*data local analysis*). O sistema suporta múltiplas escalas temporais, permitindo o ajuste da frequência do monitoramento e do tamanho da janela (de tempo). O recebimentos de dados é por *push* para monitoramento e *pull* para suporte a consultas *ad-hoc*. Foi desenvolvido um protótipo que integra-se ao hipervisor Xen.

A análise dos dados próxima da fonte pretende diminuir o volume de dados de monitoramento transmitidos. Além disso há um aumento na rapidez com que se pode obter "*insights*" a partir dos dados de monitoramento. Segundo seus autores, frequentemente é mais barato fazer uma análise rápida dos dados e enviar sumários desses, do que enviar os dados sem nenhum tratamento. Essa característica também proporciona a distribuição das tarefas de análise, aumentando o paralelismo do sistema.

O sistema possui dois tipos de componente: agentes e *brokers*. Os primeiros são responsáveis pela captura e processamento local dos dados. Os últimos realizam a correlação, agregação e análise de dados vindos de múltiplos agentes.

Quanto à arquitetura, o sistema é dividido em zonas. Essas são um veículo para o delineamento de subsistemas ou subestruturas de um *data center*, ou até mesmo múltiplos *data centers* localizados em uma mesma nuvem pública/privada. Cada zona possui um líder. Esse é escolhido entre os brokers por meio de um algoritmo de eleição. Os líderes, então, escolhem um líder entre eles e assim sucessivamente, formando um grafo inicial (hierarquia). Líderes podem iniciar uma reconfiguração de sua zona, caso a performance da agregação não seja aceitável.

Monalytics suporta descoberta, configuração e adaptação em tempo de execução. Além disso ele precisa adicionar códigos de monitoramento e análise em tempo de execução. Para isso ele utiliza um mecanismo de geração e instalação dinâmicas de código binário. Brokers e agentes têm a capacidade de realizar ações corretivas como reposta a anomalias detectadas através da análise dos dados. Dessa forma pode-se automatizar tarefas de detecção e correção de falhas.

Esse sistema possui várias semelhanças ao DIMVHCM, entre elas: monitoramento on-line, arquitetura hierárquica e utilização de *push* para o envio de dados. A grande diferença está no seu foco no gerenciamento de data centers. Daí a ênfase na análise adiantada dos dados, sem preocupar-se com a intrusividade dessa tarefa. Já no DIMVHCM se está interessado na análise do comportamento do sistema, portanto essa intrusividade é indesejável.

3.4 Considerações Finais

Nesse capítulo foram apresentados trabalhos relacionados ao proposto nessa dissertação. O primeiro trabalho visto é o TRIVA, que é uma ferramenta que implementa abordagens para a visualização de aplicações distribuídas. Após, foi apresentado o DIMVisual, que é um modelo de integração e padronização de dados de monitoramento. Essas duas ferramentas foram integradas ao DIMVHCM na implementação de um protótipo de ferramenta de monitoramento. Esse protótipo é o assunto do capítulo 5. Por último, foi apresentado um estudo de um conjunto de ferramentas de monitoramento. Esse levou em conta uma série de características, como seu objetivo, sua estrutura de distribuição de informações, armazenamento de dados históricos, integração com outras ferramentas e a apresentação das informações.

O tema do capítulo a seguir é o modelo hierárquico de coleta de dados de monitoramento proposto nessa dissertação. São apresentados seus componentes, sua arquitetura e seu funcionamento. Além disso, é feita uma comparação das ferramentas estudadas no capítulo atual em conjunto com o DIMVHCM.

4 DIMVCHM: PROPOSTA DE UM MODELO HIERÁRQUICO DE COLETA DE DADOS DE MONITORAMENTO

Conforme explicado na seção 2.1.3, uma das funcionalidades necessárias a um sistema de monitoramento distribuído é a transmissão das informações para seus usuários. Ou seja, é necessário que os dados que estão espalhados pelos recursos do sistema possam ser acessados e reunidos pelo seu utilizador. Uma das soluções lá citadas é o uso de uma estrutura hierárquica de republicadores. Sendo que cada republicador dá acesso a dados provenientes de um conjunto de componentes de nível inferior. Um exemplo de ferramenta que usa esse tipo de estrutura é o Ganglia, que foi o assunto da subseção 3.3.2.

O presente trabalho propõe um modelo hierárquico de coleta de dados de monitoramento distribuídos. O objetivo principal desse modelo é possibilitar análise *on-line* do comportamento de sistemas distribuídos, assim como da execução de programas nos mesmos. Para tornar isso possível, o modelo de coleta transmite os dados assim que tornam-se disponíveis, sem haver necessidade de esperar pelo fim do processo que está sendo observado. Assim, a ferramenta que recebe os dados do modelo pode realizar de forma *on-line* o processamento, análise e apresentação dessas informações. Versões preliminares desse modelo foram publicadas em trabalhos anteriores (TESSER; NAVAUX, 2008, 2009).

Um dos pontos interessantes sobre o modelo hierárquico é a aplicação cliente precisar de apenas um ponto de acesso ao sistema. Isso é útil em um cenário em que domínios provêm apenas uma máquina (um *frontend*) acessível externamente. Essa é uma situação bastante comum. Além disso, ele adequa-se à arquitetura utilizada em muitos sistemas distribuídos de grande escala, em que os componentes são agrupados de forma hierárquica. A figura 4.1 demonstra essa característica. Nela temos, à esquerda, uma representação na forma de conjuntos em que: um *Grid* contém *sites* (ou domínios); um *site* contém *clusters*; e um *cluster* contém nós. Já do lado direito tem-se isso representado de forma hierárquica em uma árvore, na qual ramos são subordinados ao nó pai. Por questão de espaço são omitidos da árvore os nós de um dos clusters. Esse esquema pode ser aprofundado para níveis inferiores, como processadores e *cores* ou processos e threads.

O modelo proposto é composto de três tipos de componentes: coletores, agregadores e clientes. Os coletores são responsáveis por obter os dados localmente aos recursos e transmiti-los a um ou mais agregadores. Os agregadores fazem o papel de republicadores, dando acesso a dados provenientes de um conjunto de componentes de nível inferior. Pode-se formar uma hierarquia de agregadores em que agregadores de um determinado nível republicam os dados vindos de outros, de nível hierárquico inferior. Os clientes são responsáveis por receber dados de agregadores e repassá-los a ferramentas, para que essas possam realizar o processamento, análise e apresentação das informações coletadas.

Os dados de monitoramento são propagados através da hierarquia usando o modelo

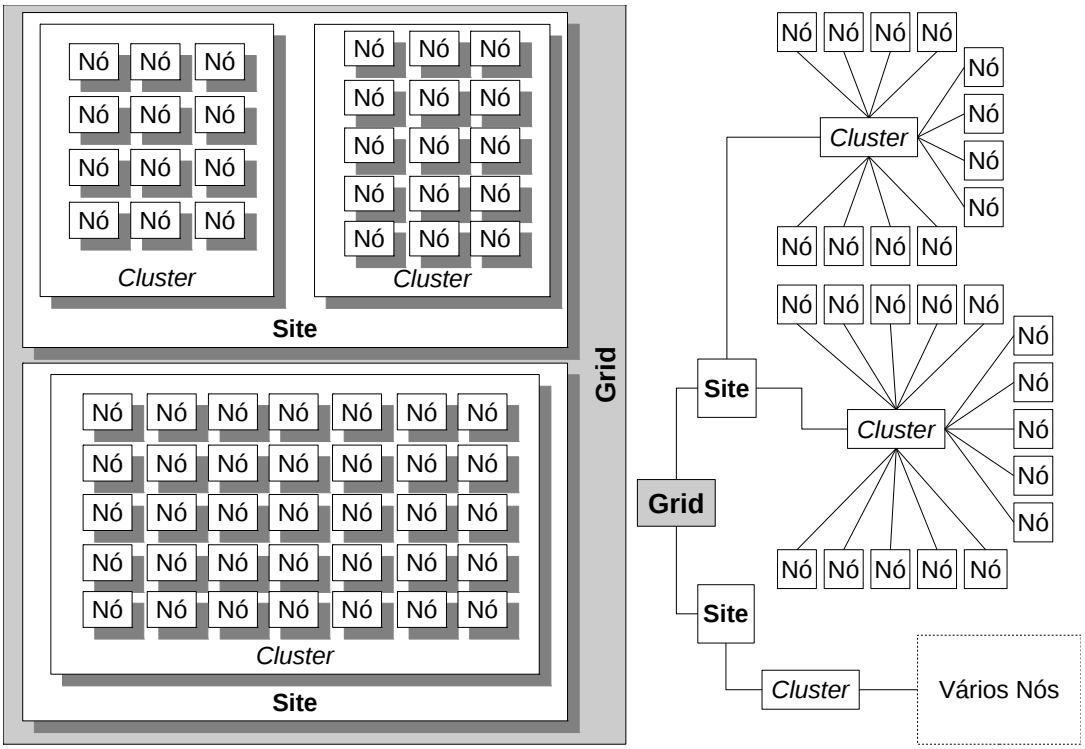


Figura 4.1: Grid (multicluster) visto como conjunto e como hierarquia.

push de recebimento de dados. O modelo *push* é mais adequado ao monitoramento (CZAJKOWSKI et al., 2001), visto que deseja-se acompanhar as mudanças no estado dos objetos durante o período observado. Pelo uso de *push* elimina-se a necessidade do envio das requisições de dados. Por estarmos interessados em, dentro do possível, observar todas as mudanças de estado, não há necessidade do modelo sob demanda.

Nem sempre é necessário utilizar todas as informações de monitoramento geradas. Pode ser que para compreender o fenômeno que se quer observar só seja preciso um subconjunto dessas. Por isso, é interessante que se possa restringir ou escolher que dados serão recebidos. Dessa maneira, diminui-se o uso dos recursos do sistema através da diminuição da transmissão de dados inúteis. O modelo de coleta aqui proposto possui um mecanismo de filtragem através do uso de subscrição. Esse permite que um programa cliente escolha de quais coletores deseja receber dados. Ou seja, os coletores e agregadores só transmitem os dados aos componentes registrados para recebê-los.

Neste trabalho focou-se na utilização de uma ferramenta de visualização para a análise do comportamento de aplicações. Primeiramente os dados são fornecidos ao DIMVisual (SCHNORR; NAVAUX; OLIVEIRA STEIN, 2006), que é um modelo de integração de dados de monitoramento. Esse modelo, que foi o assunto da seção 3.1, gera dados integrados e convertidos para o formato de entrada de uma ferramenta de visualização. Detalhes sobre como foi feita a integração entre modelo hierárquico, DIMVisual e a ferramenta de visualização, são apresentado no capítulo 5, que trata da implementação do protótipo que foi utilizado para avaliar o modelo aqui proposto. Devido à integração com o DIMVisual, decidiu-se nomear esse modelo "*DIMVisual Hierarchical Collection Model*" (DIMVHCM).

Na seção a seguir são apresentados mais detalhadamente os componentes do modelo de coleta. Na seção 4.2 explica-se o processo de registro dos coletores que é necessário para a descoberta e subscrição a estes. Na 4.3 é explicado o funcionamento do mecanismo

de subscrição. A seguir, na 4.4, é abordado o processo de transmissão dos dados através da hierarquia. Na seção 4.5 é mostrado como os componentes do modelo interagem na realização do monitoramento. Após isso, na seção 4.6, é comparado um conjunto de ferramentas de monitoramento constituído pelas ferramentas analisadas no capítulo 3 mais o DIMVHCM. Finalmente, na seção 4.7 são exploradas algumas possibilidades de aplicação e evolução do modelo.

4.1 Componentes

A figura 4.2 exemplifica distribuição dos componentes do DIMVHCM em um sistema distribuído fictício. Essa figura representa dois *clusters*: um com *n* e outro com *k* nós. Cada nó possui três tipos de coletores: *C1*, *C2* e *C3*. Cada um desses provê acesso a dados gerados por uma ferramenta de monitoramento, conforme mostrado no primeiro nó do *cluster 1*. Nota-se que o coletor do tipo *C3* está integrado na ferramenta chamada *Tool 3*. Isso indica que essa ferramenta utiliza um componente coletor internamente, sendo desenvolvida especificamente para ser usada com o DIMVHCM. Os coletores fornecem acesso aos dados locais a cada nó, através de sua transmissão a um agregador acessível externamente ao *cluster*. Os dados dos agregadores situados em cada um dos *clusters* são enviados para um agregador de nível superior (*Agregador 3*). Através desse agregador os clientes podem receber as informações de monitoramento de ambos os *clusters* que compõe o sistema.

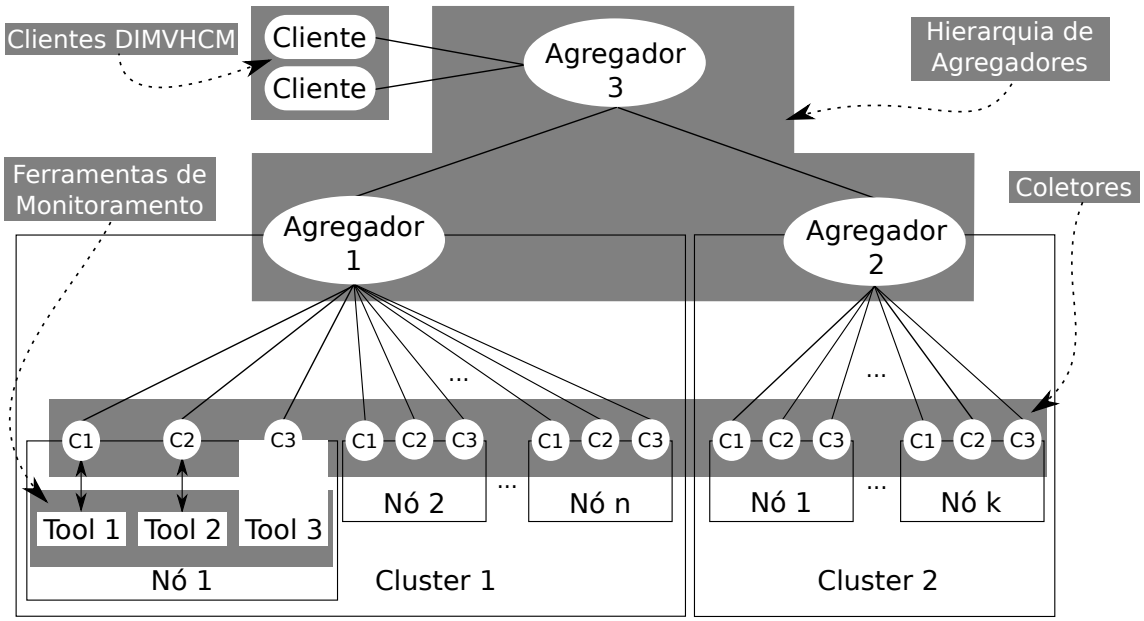


Figura 4.2: Componentes do Modelo Hierárquico de Coleta de Dados de Monitoramento

Nas subseções a seguir serão apresentados detalhes de cada tipo de componente que compõe o DIMVHCM. São abordadas sua inicialização e as funções pelas quais cada cada um deles é responsável.

4.1.1 Coletores

Coletores são componentes responsáveis por obter informações de monitoramento geradas por outras ferramentas ou pela execução de programas instrumentados. Alterna-

tivamente pode-se escrever um programa que gere as informações de monitoramento e utilize o coletor internamente. Coletores executam localmente ao recurso observado.

Na sua inicialização os coletores registram-se em um conjunto de agregadores, fornecendo informações como um identificador único e o seu tipo (ver seção 4.2). Esse tipo identifica quais informações ele fornece e o formato no qual elas estão. A implementação do coletor é dependente da fonte de onde ele obtém as informações de monitoramento.

O coletor envia os dados obtidos para os agregadores que possuem subscrição para receber seus dados. Isso é feito toda vez que uma condição é satisfeita, por exemplo: a acumulação de dados de um certo número de leituras na fonte; a decorrência de um determinado intervalo de tempo desde o último envio.

4.1.2 Agregadores

Os agregadores servem como ponto de acesso a dados de um conjunto de componentes de nível inferior. Dessa maneira, eles agem com republicadores de dados. A implementação dos agregadores é independente da fonte de dados.

Vale ressaltar que, nessa primeira versão do DIMVHCM, os agregadores não processam os dados. A agregação refere-se apenas à integração do acesso a dados de várias origens. Em futuras versões pretende-se adicionar a capacidade de preprocessar os dados recebidos, antes de repassá-los.

Agregadores podem receber dados tanto de coletores como de outros agregadores. Os dados recebidos são enviados para componentes de nível superior que possuam subscrição aos dados provenientes do coletor que os transmitiu inicialmente. Os componentes que recebem esses dados podem ser agregadores ou clientes. Além disso, os agregadores repassam informações de registro dos coletores até o topo da hierarquia e requisições de subscrição feitas por clientes até os coletores (ver subseção 4.3).

4.1.3 Clientes

Os clientes são responsáveis por receber dados de um conjunto de agregadores e repassá-los para um componente capaz de processar essas informações. Os clientes proporcionam ao usuário a capacidade de associar um objeto receptor a cada tipo de coletor. Ao receber dados de monitoramento, o cliente verifica o tipo do coletor que os gerou e os repassa ao objeto associado.

Cada um desses receptores deve ser capaz de processar dados fornecidos pelo tipo do coletor ao qual está associado. Ou seja, ele deve conhecer o conjunto de informações recebidas e seu formato. Dessa maneira, os receptores podem utilizá-las para realizar alguma tarefa. Na seção 1.1, foram citados alguns exemplos de tarefas que poderiam ser realizadas por esses objetos. O cliente desenvolvido neste trabalho repassa as informações para fontes de dados do DIMVisual.

Além disso, o cliente permite registrar um gerenciador de subscrições (ver seção seguinte). O cliente repassa os registros de coletores a esse componente. O gerenciador possibilita escolher a quais coletores subscrever-se, baseado nessas informações de registro.

4.2 Registro de Coletores

Quando um coletor é inicializado, é necessário que ele registre-se em um conjunto de agregadores. Esses agregadores são os únicos que irão receber dados diretamente daquele coletor. Esse registro é repassado pelos agregadores até o topo da hierarquia, sendo que

todos os agregadores no caminho passarão a ter o coletor registrado.

Os dados fornecidos pelo coletor nesse processo são uma identificação única, seu tipo e sua localização. A identificação única será usada pra identificar as informações relacionadas ao coletor, em cada componente. O tipo é usado pelo cliente para identificar quais as informações fornecidas pelo coletor. A localização permite saber onde as informações estão sendo coletadas.

Agregadores guardam um registro dos coletores dos quais podem receber dados. Nesse registro ficam também indicados os componentes que estão subscritos para receber dados daquele coletor através do agregador. Os agregadores repassam esse registro para os componentes de nível superior a que estão conectados. Juntamente com cada registro é passada uma referência a si mesmo. Essa informação é utilizada no processo de subscrição.

Quando um cliente conecta-se em um agregador ele obtém os registros dos coletores acessíveis através desse. Essa informação permite a descoberta dos coletores disponíveis, das informações que eles fornecem e a quais recursos elas referem-se (localização). Caso um novo coletor seja adicionado, após o registro do cliente, a propagação desse registro pela hierarquia fará com que o cliente também seja atualizado. Caso haja um gerenciador de subscrições registrado, o cliente repassa os registros de coletores para ele.

4.3 Subscrição

Para diminuir a transmissão de dados desnecessários aos clientes, o DIMVHCM possui um mecanismo de subscrição. Esse pode ser considerado um mecanismo de filtragem de distribuição, que fornece filtragem implícita das informações, conforme visto na subseção 2.1.2.

No DIMVHCM os clientes podem escolher de quais coletores desejam receber informações de monitoramento. Caso se deseje um controle mais fino sobre quais dados serão transmitidos, pode-se criar um maior número de coletores mais especializados. Por exemplo, cada coletor poderia ser responsável por enviar um subconjunto dos dados obtidos de uma determinada fonte. A seguir, descreve-se com mais detalhes o funcionamento do mecanismo de subscrição.

O cliente do DIMVHCM pode registrar um componente como gerenciador de subscrições. Esse receberá notificações quando novos coletores forem registrados ou removidos. O gerenciador pode fornecer essas informações a um usuário que decidirá a quais coletores subscrever-se. Esse usuário pode ser tanto uma pessoa, através de uma interface interativa, quanto um programa.

Os clientes possuem acesso a informações de localização e ao tipo de cada coletor. Essas são fornecidas quando os coletores registram-se e são repassadas nas notificações ao gerenciador de subscrições. Cada tipo de coletor deve estar associado diretamente a um único formato de dados e a um único conjunto de informações de monitoramento. Por exemplo, o tipo *GangliaCPU* poderia estar associado a um coletor que obtém informações de um *daemon* do Ganglia, sobre o uso de CPU no nó em que está localizado. É necessário que o responsável por decidir quais dados necessitam ser recebidos tenha conhecimento do significado de cada tipo. Baseado nessa informação e nos recursos que se quer monitorar, pode-se escolher um conjunto de coletores dos quais deseja-se receber dados. Então, basta solicitar ao cliente que se subscreva a esses coletores. Da mesma maneira, pode-se solicitar o cancelamento da subscrição a um coletor, quando não se quiser mais receber dados desse.

Quando um agregador recebe uma requisição de subscrição, esse registra o requerente como subscritor do coletor pedido. Então ele verifica se ele mesmo já possui subscrição daquele coletor. Caso negativo, da mesma maneira que o cliente, ele faz uma solicitação de subscrição para o componente adequado, do nível logo abaixo a ele. O receptor dessa requisição pode ser tanto outro agregador como um coletor. Vale a pena ressaltar que para o componente de nível inferior, o subscritor será o agregador e não o cliente que iniciou o processo de subscrição. Não há necessidade que seja conhecido o destino final dos dados. Um componente só precisa saber para quem ele precisa enviá-los. Quando um coletor recebe uma requisição de subscrição, ele apenas registra o componente que realizou a requisição na sua lista de subscritos.

Um componente do DIMVHCM só enviará os dados de monitoramento para componentes que tiverem registradas subscrições do coletor que primeiramente os enviou. A transmissão de dados através da hierarquia é o assunto da seção seguinte.

4.4 Transferência de dados

O DIMVHCM é responsável por transmitir os dados reunidos nos coletores para os clientes que possuem subscrição a esses. Essa transferência ocorre da seguinte forma:

1. Um coletor obtém dados de monitoramento gerados por uma ferramenta externa ou por uma aplicação que tenha sido instrumentada para gerá-los. A maneira como esse processo é realizado é dependente de cada fonte de dados.

2. O coletor verifica quais os agregadores para os quais tem subscrições registradas e envia os dados coletados para eles.

3. Quando um agregador recebe dados de monitoramento de um coletor, ele verifica no seu registro interno quais componentes de nível superior possuem a subscrição para aquele coletor. Então ele repassa os dados recebidos para esses componentes. Esse processo repete-se em todos os agregadores que receberem os dados, até que todos os clientes subscritos àquele coletor recebam as informações.

4. Quando um cliente recebe dados de monitoramento, ele verifica o tipo do coletor de origem. Usando dessa informação ele descobre o componente responsável por processar aqueles dados. Feito isso, as informações são repassadas ao componente adequado.

4.5 Monitoramento utilizando o DIMVHCM

Essa seção apresenta uma visão geral da organização dos componentes do DIMVHCM em um sistema de monitoramento. A figura 4.3 ilustra uma possível estrutura de monitoramento, assim como interações entre os componentes. No nível inferior temos três exemplos de uso de coletores. À esquerda (Coletor tipo 1), temos um coletor que obtém dados de uma ferramenta de monitoramento externa. No meio, temos programa instrumentado que repassa informações de monitoramento para um coletor instanciado dentro do próprio programa. Já no exemplo mostrado à direita, um coletor é utilizado internamente por uma ferramenta que obtém dados de monitoramento.

Acima dos coletores vê-se dois níveis de agregadores e no nível superior tem-se uma aplicação cliente construída utilizando um cliente do DIMVHCM. Setas são utilizadas

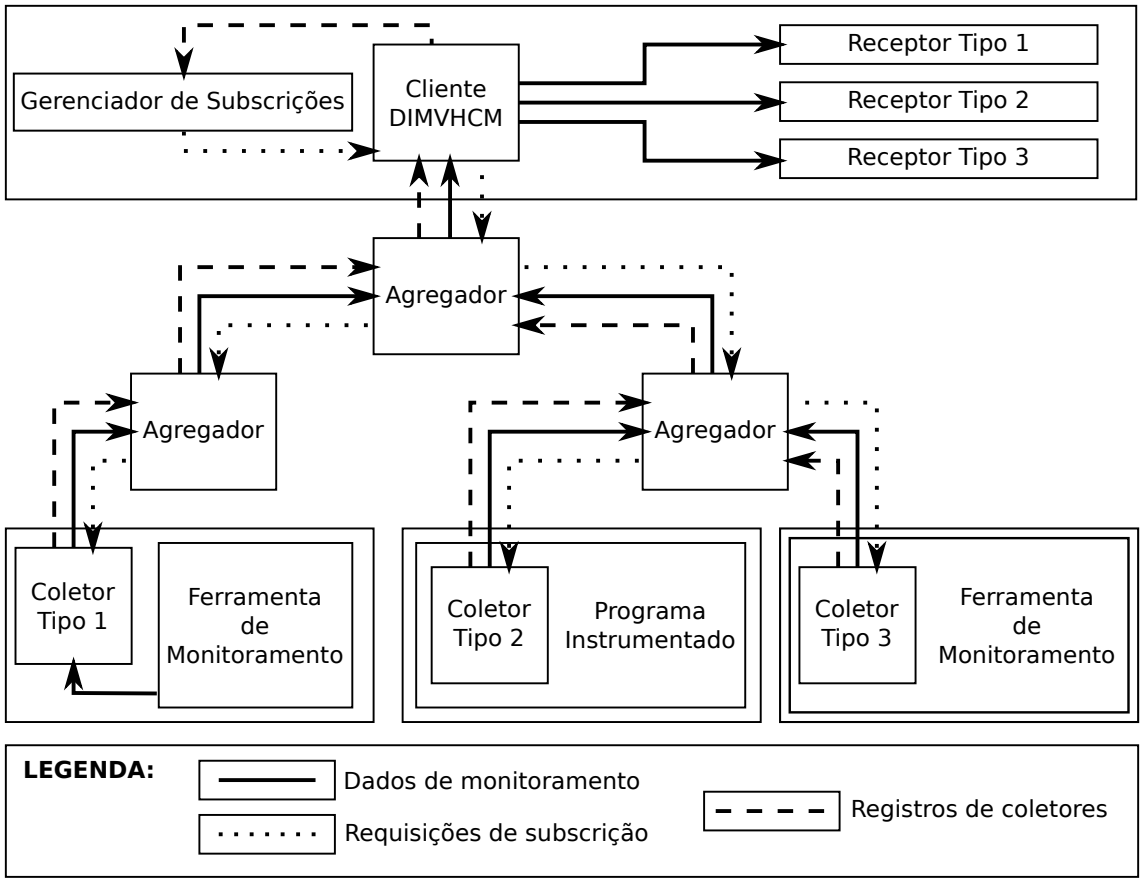


Figura 4.3: Exemplo de organização dos componentes do DIMVHCM em um sistema de monitoramento.

para simbolizar o fluxo de informações entre os componentes. Dados de monitoramento e registros de coletores são propagados para cima na hierarquia. Solicitações e cancelamentos de subscrição são propagados para baixo.

No cliente os dados de monitoramento são encaminhados para receptores específicos para cada tipo de cliente. Cada um desses receptores implementa o processamento de um formato de dado. Os registros de coletores são repassados ao gerenciador de subscrições. Esse gerenciador permite a seleção dos agregadores aos quais o cliente deve subscrever-se. Esse processo de decisão pode ser interativo ou automatizado. Uma vez decida a subscrição a um coletor, o gerenciador deve solicitá-la ao cliente DIMVHCM. Esse último irá repassar a solicitação à hierarquia de agregadores. O processo de propagação dessa solicitação na hierarquia foi explicado na seção 4.3.

4.6 Comparação de Ferramentas de Monitoramento

Nesta seção é apresentada uma comparação entre ferramentas de monitoramento, incluindo as que foram assunto da seção 3.3 e o DIMVHCM. São levadas em conta características relacionadas à distribuição, armazenamento e apresentação das informações de monitoramento. Também é apresentada a classificação de cada uma delas, segundo a taxonomia de Zanikolas e Sakellariou, abordada na seção 2.4.3.

Quanto à distribuição de informações de monitoramento, a tabela 4.1 mostra uma comparação de acordo com quatro características. A primeira característica é o suporte

aos modelos *pull* e *push* de recebimento de dados. A ferramenta Ganglia utiliza o modelo *push* internamente a um domínio administrativo, através do uso de *multicast* pelos *gmond*. Já os *gmetad* utilizam um modelo *pull* para receber os dados descobertos por *polling* pelos membros da hierarquia de republicadores. Enquanto isso, a ferramenta GridICE suporta apenas *pull* e a MonALISA suporta apenas *push*. O Globus MDS versão 2 suporta apenas o modelo *pull*, enquanto a versão 4 suporta ambos modelos. O Monalytics suporta *push*, para monitoramento, e *pull* para possibilitar consulta *ad-hoc*. O DIMVHCM utiliza um modelo *push* de recebimento de dados. A segunda característica é o suporte ao modelo *publish / subscribe*, que é suportado pelas ferramentas *MonALISA*, Globus MDS versão 4 e Monalytics, dentre as analisadas. Como essas, o DIMVHCM possui um mecanismo de *publish / subscribe*. A terceira propriedade comparada é a existência de republicadores. Ganglia, MDS e Monalytics formam uma hierarquia de republicadores, assim como o DIMVHCM, no qual esses são chamados agregagores. GridICE utiliza um republicador por *site* monitorado e MonALISA não utiliza republicadores para as informações de monitoramento. A última característica, com relação à distribuição, diz respeito ao esquema usado para descoberta da disponibilidade de informações de monitoramento. Ganglia utiliza um *heartbeat*, ou seja, o envio periódico, pelos *gmond*, de uma mensagem indicando sua disponibilidade. O GridICE utiliza MDS-2 para descoberta. O MonALISA utiliza o mecanismo de *lease* do JINI LUS. O MDS versão 2 utiliza LDAP para manter as informações utilizadas para descoberta. Na versão 4, os *information sources* devem registrar-se periodicamente nos *aggregator services* adequados, utilizando *Web Services*. Monalytics suporta descoberta em tempo de execução. No entanto, os autores não especificam o mecanismo utilizado para implementá-la. No DIMVHCM os clientes podem descobrir os coletores através da hierarquia de agregadores.

Tabela 4.1: Comparação das ferramentas em relação à distribuição das informações de monitoramento

	pull/push	**publish/ subscribe**	**Republicadores**	**Descoberta**
Ganglia	gmond: push gmetad: pull	Não	Hierarquia	Heartbeat
GridICE	pull	Não	1/site	MDS-2
MonALISA	push	Sim	Não	JINI LUS
MDS 2	pull	Não	Sim	LDAP
MDS 4	ambos	Sim	Sim	Web Services
Monalytics	Ambos	Sim	Sim	Sim
DIMVHCM	push	Sim	Sim	Hierarquia de Agregadores

No que se refere ao armazenamento de informações de monitoramento, foram comparadas duas características, conforme mostra a tabela 4.2. Uma propriedade levada em conta é o fator limitante do tempo de vida dos dados. Na Ganglia isso depende do tamanho do arquivo usado pelo RRDtool. Quanto às ferramentas GridICE e MonALISA, não foram encontradas informações sobre esse aspecto. O MDS versão 2 não possui um serviço de arquivamento próprio. Na versão 4, os clientes especificam por quanto tempo necessitam da informação. Também foi considerada a utilização, ou não, de alguma técnica de sumarização das informações mais antigas. Na Ganglia, a sumarização é feita através do uso da RRDtool. A ferramenta MonALISA guarda as médias e o intervalo de flutuação

dos valores mais antigos. Não encontrou-se informações sobre a ferramentas GridICE e MDS versão 4, quanto a essa característica. Monalytics não especifica essas duas características. No entanto, seu funcionamento pode ser modificado pela inserção dinâmica de código. Portanto, seria possível implementar diferentes abordagens, conforme a necessidade. O DIMVHCM por si só, não possui nenhum controle sobre o armazenamento dos dados. Portanto para essa análise, foi considerado o cliente implementado, utilizando DIMVisual e TRIVA. Mais detalhes sobre esse cliente são apresentados no capítulo 5, que trata da implementação de um protótipo do DIMVHCM. Esse último mantém os dados em memória durante a execução. Isso é feito para permitir a navegação nos dados com relação ao tempo. Além disso, os dados antigos são armazenados sem sumarização.

Tabela 4.2: Comparação das ferramentas em relação ao armazenamento de dados históricos

	Tempo de Vida	**Sumarização**
Ganglia	Limitado pelo tamanho do arquivo	Sim (RRDtool)
GridICE	?	?
MonALISA	?	Médias
MDS 2	N. A.	N. A.
MDS 4	especificação do cliente	?
Monalytics	depende de implementação	depende da implementação
DIMVHCM	durante a execução	Não

Quatro características foram levadas em conta em relação à apresentação das informações de monitoramento, conforme mostra a tabela 4.3. É importante ressaltar que o Globus MDS versão 2 e o Monalytics não possuem interfaces próprias de apresentação visual de informações. Quanto à versão 4 do MDS, foi considerada a interface WebMDS. Um dos aspectos analisados foi a possibilidade de visualizar as informações em vários níveis de abstração. Outra, foi a possibilidade de navegação com relação ao tempo, ou seja, voltar e avançar a visualização de maneira a observar o que acontece em determinados intervalos de tempo. Nenhuma das ferramentas apresentadas no capítulo 3 possui qualquer dessas duas capacidades. A terceira característica comparada é a possibilidade de selecionar um grupo de informações a serem visualizadas, inclusive quais objetos quer-se observar. Nesse caso, apenas a GridICE possibilita alguma forma de seleção. A última característica, nessa categoria, é a exibição de interações entre os objetos observados. Essas interações poderiam ser, por exemplo, comunicações entre processos ou entre nós de um sistema distribuído. Novamente, nenhuma das ferramentas analisadas implementa essa funcionalidade. Como dito anteriormente, o protótipo desenvolvido utiliza o TRIVA como componente de visualização. Mais especificamente a visualização utilizando *treemap*. Portanto, no que diz respeito à visualização com o DIMVHCM, foram consideradas as características do TRIVA. Essas foram apresentadas na seção 3.2. Esse suporta a visualização em diversos níveis de abstração, pois possibilita a visualização sumarizada para diferentes níveis da hierarquia física do sistema. Ou seja, em diferentes níveis de abstração. O TRIVA também possibilita a navegação com relação ao tempo. Portanto, através da integração com o TRIVA, adiciona-se ao DIMVHCM duas características que não estão presentes em nenhuma das ferramentas estudadas. Além disso, uma interface gráfica possibilita ao usuário selecionar quais entidades da hierarquia serão visualizadas. Essa característica está presente em apenas uma das ferramentas estudadas no capítulo anterior. A visualização do tipo *treemap* não suporta a exibição de interações entre entidades. Por-

tanto, foi colocado um "Não"na tabela, apesar de o TRIVA suportar a visualização dessas interações em outros tipos de visualização. É interessante notar que as outras ferramentas estudadas também não possibilitam a visualização dessas interações.

Tabela 4.3: Comparação das ferramentas em relação à apresentação das informações de monitoramento

	Vários níveis de abstração	Navegação em relação ao tempo	Seleção de informações	Mostra interações entre objetos
Ganglia	Não	Não	Não	Não
GridICE	Não	Não	Sim	Não
MonALISA	Não	Não	Não	Não
MDS 2	N. A.	N. A.	N. A.	N. A.
MDS 4	Não	Não	Não	Não
Monalytics	N. A.	N. A.	N. A.	N. A.
DIMVHCM	Sim	Sim	Sim	Não

A tabela 4.4 mostra a classificação das ferramentas estudadas, segundo a taxonomia de Zanikolas e Sakellariou. O Ganglia é um sistema L3.G, pois possui uma hierarquia de republicadores (*gmetad*) arbitrariamente distribuída e é genérico quanto à natureza das entidades monitoradas. GridICE possui uma estrutura centralizada, e não possui uma API de produtores. Além disso, ele é um sistema genérico no que diz respeito ao tipo de entidade monitorada. Esse sistema também tem a capacidade de utilizar publicadores e republicadores de outras ferramentas. Portanto, GridICE é um sistema L0.G.S, ou seja, de nível 0, propósito genérico e empilhável. MonALISA é um sistema L3.G.S, pois permite a formação de uma hierarquia de republicadores arbitrariamente distribuída, é genérico quanto às entidades monitoradas e é capaz de utilizar publicadores e republicadores de outras ferramentas. O Globus MDS versão 2 é um sistema L3.G.S. Com essa ferramenta pode-se formar uma hierarquia de republicadores, que usam os protocolos GRIP e GRRP (conforme foi visto na seção 3.3.5), distribuídos arbitrariamente. Além disso, o sistema não é destinado ao monitoramento de tipos específicos de entidades e é empilhável. O MDS versão 4 também possui essa classificação (L3.G.S), diferindo nos mecanismos usados para a propagação dos dados (Web Services). O Monalytics é um sistema genérico quanto às atividades monitoradas. Além disso, pode ser considerado empilhável, devido à possibilidade de adicionar códigos para interação com outras ferramentas. Ele é um sistema de nível 3, devido à possibilidade construir uma hierarquia de republicadores distribuídos arbitrariamente. Juntando essas características, o Monalytics obtém a classificação L3.G.S. na taxonomia. O DIMVHCM é um sistema de nível 3, pois os agregadores formam uma hierarquia de republicadores distribuída arbitrariamente. Além disso, ele é genérico quanto às entidades monitoradas. A interação com outras ferramentas pode ser feita através de coletores e fontes de dados implementados para essa tarefa. Portanto, o DIMVHCM pode ser considerado empilhável. Então, a classificação do DIMVHCM é L3.G.S, segundo essa taxonomia.

4.7 Outras possibilidades

Nessa seção serão apresentados alguns aspectos interessantes do modelo hierárquico que não foram exploradas nesse trabalho por estarem fora de seu foco principal. Primeiro é abordada a possibilidade de uso de componentes redundantes e os prováveis benefícios

Tabela 4.4: Comparação das ferramentas segundo classificação na taxonomia de Zanikolas e Sakellariou

	Nível	Entidades monitoradas	Empilhável	Classificação
Ganglia	3	Genérico	Não	L3.G
GridICE	0	Genérico	Sim	L0.G.S
MonALISA	3	Genérico	Sim	L3.G.S
MDS 2	3	Genérico	Sim	L3.G.S
MDS 4	3	Genérico	Sim	L3.G.S
Monalytics	3	Genérico	Sim	L3.G.S
DIMVHCM	3	Genérico	Sim	L3.G.S

de fazê-lo.

O modelo proposto permite uma grande flexibilidade na construção da hierarquia de componentes. Um exemplo disso é a possibilidade de mais de um agregador receber dados de uma mesma origem. Isso permite a exploração de aspectos relacionados ao uso de componentes redundantes. No entanto, para fazer uso dessa redundância seria necessário adicionar ao modelo técnicas para a seleção de réplicas.

Um exemplo de uso seria a construção de um mecanismo de tolerância a falhas. Para isso, poder-se-ia ter agregadores de reserva, para os quais só se enviaria dados em caso de falha de um agregador principal. Nesse caso, tanto o agregador principal, quanto os de reserva estariam conectados aos mesmos componentes de nível superior. Lembrando que os componentes que recebem os dados não precisam saber quem os enviou. A única necessidade na troca de agregador seria modificar a informação que diz para quem enviar as solicitações de subscrição.

Uma outra possibilidade seria ter-se agregadores separados para conjuntos diferentes de informações. Com isso poder-se-ia restringir os dados a que cada usuário ou grupo de usuários tem acesso. Isso poderia ser feito através de ferramentas externas que restrinjam o acesso aos agregadores. Dessa forma ter-se-ia diferentes hierarquias, com acesso a diferentes informações, dependendo do nível de autorização que o usuário possui.

4.8 Considerações Finais

O *DIMVisual Hierarchical Collection Model* (DIMVHCM), que foi o assunto deste capítulo, é um modelo para coleta *on-line* de dados de monitoramento de sistemas distribuídos. Seu objetivo principal é possibilitar a análise *on-line* do comportamento de sistemas e aplicações distribuídos.

Para atingir esse objetivo o modelo utiliza coletores locais, uma hierarquia de agregadores distribuídos e clientes. O coletores obtém os dados e os enviam a agregadores. Esses, então, repassam-nos através da hierarquia até chegarem aos clientes. Os clientes repassam os dados a componentes capazes de processá-los. O DIMVHCM utiliza um modelo *push* de recebimento de dados. Além disso, ele inclui um mecanismo de subscrição. Através desse, os clientes podem selecionar de quais coletores desejam receber dados.

Nesse capítulo foi apresentado o funcionamento dos componentes, assim como os processos de registro de coletores e gerenciamento de subscrições. Também foi mostrado como os componentes interagem para formar um sistema de monitoramento. Além disso, foi apresentada uma comparação do de ferramentas de monitoramento, incluindo as estudadas no capítulo anterior e o DIMVHCM.

No capítulo seguinte, será abordada a implementação de um protótipo de ferramenta de monitoramento, utilizando o DIMVHCM. Para isso, ele foi integrado ao DIMVisual, para integração e conversão de dados, e ao TRIVA, para sua visualização.

5 IMPLEMENTAÇÃO DO PROTÓTIPO

Para avaliar o modelo, foi realizada a implementação de um protótipo. Esse consiste da integração de três componentes: DIMVHCM, DIMVisual e TRIVA. A figura 5.1 ilustra de maneira simplificada o fluxo de dados entre os componentes. O componente `DIMVHCMReader`, que aparece na figura, é um módulo do TRIVA, que foi desenvolvido para "conectá-lo" ao DIMVisual. Seguindo o que mostra a figura, o DIMVHCM fornece os dados coletados nos recursos ao DIMVisual. Esses dados estão no formato em que foram obtidos de quem os gerou. Então, o DIMVisual integra e padroniza os dados recebidos. O resultado disso são dados no formato da ferramenta de monitoramento Pajé, que são obtidos pelo módulo `DIMVHCMReader` do TRIVA. Esse módulo controla a entrada dos dados na ferramenta, que gera como resultado final uma visualização.

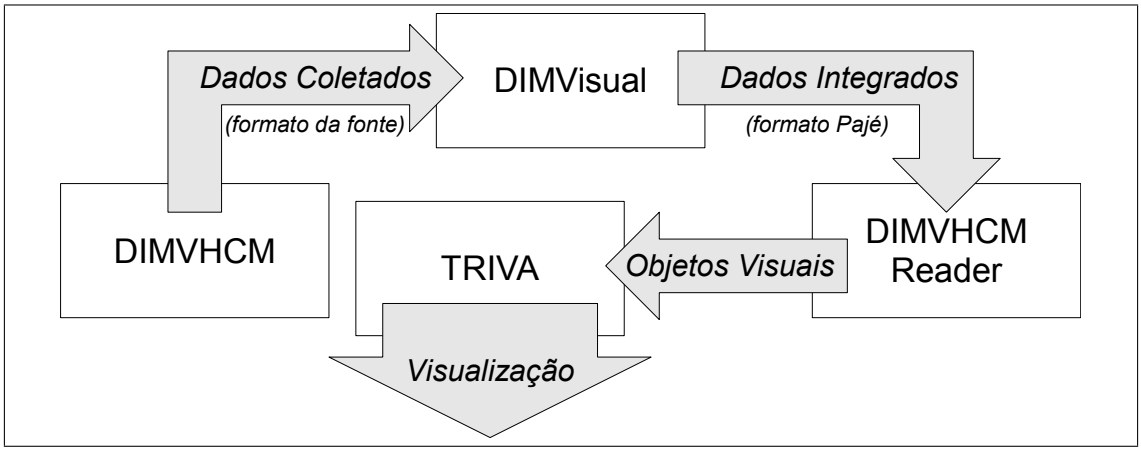


Figura 5.1: Componentes do Protótipo Implementado

É válido ressaltar que todo esse processo é realizado *on-line*, ou seja, os dados são coletados, em tempo de execução, conforme são gerados e a visualização é atualizada quando da chegada de novas informações de monitoramento. Para isso foi necessária a adaptação dos protótipos do DIMVisual e do TRIVA, visto que esses suportavam apenas a visualização *post-mortem*. Ambos funcionavam através da leitura de arquivos de rastros reunidos previamente pelo usuário.

O protótipo foi desenvolvido na linguagem Objective-C. Essa é a mesma linguagem em que foram implementados o protótipo do DIMVisual e o TRIVA. Mais especificamente, foram utilizadas as bibliotecas do projeto GNUstep, que tem por objetivo prover uma implementação livre e aberta dos API's e ferramentas do framework Cocoa, para várias plataformas.

Esse framework inclui API's para a implementação de objetos distribuídos. Ele tam-

bém fornece um servidor de nomes através do qual os objetos podem registrar-se e localizar outros objetos. Para poder fazer chamadas a um objeto remoto é necessário obter um proxy. Esse pode ser obtido através de busca no servidor de nomes ou através de recebimento de referência de um objeto remoto. Por exemplo: um coletor conecta-se a um agregador que ele localizou através do servidor de nomes; então ele faz uma chamada remota ao agregador, passando como argumento uma referência a si mesmo; assim o agregador pode fazer chamadas ao coletor, utilizando essa referência. Nesse exemplo é interessante notar que o agregador não precisou saber o nome do coletor.

As chamadas a objetos remotos possuem a mesma sintaxe que as locais. Não há a necessidade de possuir localmente a classe do objeto remoto. Para que um objeto conheça quais os métodos que podem ser executados no objeto remoto, ele precisa de um protocolo, que é similar ao que algumas linguagens chamam de interface. Ele contém protótipos de métodos que o objeto remoto implementa. O conhecimento do protocolo também ajuda na detecção de erros sintáticos em tempo de compilação. Além disso, ele pode conter outras informações como: se um parâmetro deve ser passado por cópia ou por referência; se modificações nos objetos passados como argumentos devem ser repassadas ao chamador do método. No caso de argumentos passados por cópia, é necessário que suas classes implementem o protocolo `NSCoding`. Esse é o caso dos dados de monitoramento no DIMVHCM.

Para receber chamadas remotas, um objeto deve estar executando um laço, chamado `RunLoop`, que é iniciado através de uma chamada de método. Caso seja necessário que o programa execute outras tarefas ao mesmo tempo que está esperando por essas chamadas, então é necessário que esse laço seja executado em uma thread separada. Por exemplo, os coletores precisam ser capazes de receber solicitações de subscrição ao mesmo tempo que estão coletando os dados. Um outro detalhe importante é que o `RunLoop` deve executar na mesma thread em que foram criadas as conexões. Isso faz com que as chamadas remotas tenham que ser executadas nessa mesma thread. Esse é o caso de quando um coletor tem que enviar dados para um objeto remoto ou um cliente tem que enviar uma solicitação de subscrição. A API possui métodos que possibilitam a execução de métodos em threads diferentes. Processar essas chamadas vindas de diferentes threads é uma outra função do `RunLoop`.

Na seção a seguir apresentam-se detalhes da implementação do DIMVHCM. Incluem-se informações sobre classes desenvolvidas para dar suporte à criação de coletores, agregadores e clientes personalizados. Na seção 5.2, é abordada a implementação do cliente. Essa inclui, entre outros, a adaptação dos protótipos do DIMVisual e do TRIVA para dar suporte ao monitoramento *on-line*.

5.1 DIMVisual Hierarchichal Collection Model

A implementação do DIMVHCM constou inicialmente do desenvolvimento de classes de objetos distribuídos que auxiliam na implementação dos componentes do modelo. Essas classes implementam o gerenciamento de conexões; o envio de mensagens entre os componentes; a manutenção informações de registro de coletores e subscrições.

As classes são chamadas `DIMVCollector`, `DIMVAggregator`, `DIMVClient`. A figura 5.2 mostra-as em um exemplo de hierarquia. Nela pode-se ver as classes sendo utilizadas para implementar dois tipos de coletor (*ColetorTipo1* e *ColetorTipo2*); como parte principal dos agregadores e para fornecer dados para um integrador, em um cliente.

As três classes possuem alguns argumentos de inicialização em comum, que são: um

nome e um identificador únicos e uma lista de nomes de agregadores aos quais conectar-se. O nome ajuda na identificação a objetos remotos; o identificador é usado nos registros mantidos por cada componente do DIMVHCM; os nomes de agregadores são usados para localizá-los utilizando um servidor de nomes. Na presente implementação apenas os agregadores precisam ser localizados dessa maneira. Os outros componentes são acessíveis através de referências a seus objetos. Essas são passadas aos agregadores numa chamada de registro, realizada logo após à conexão.

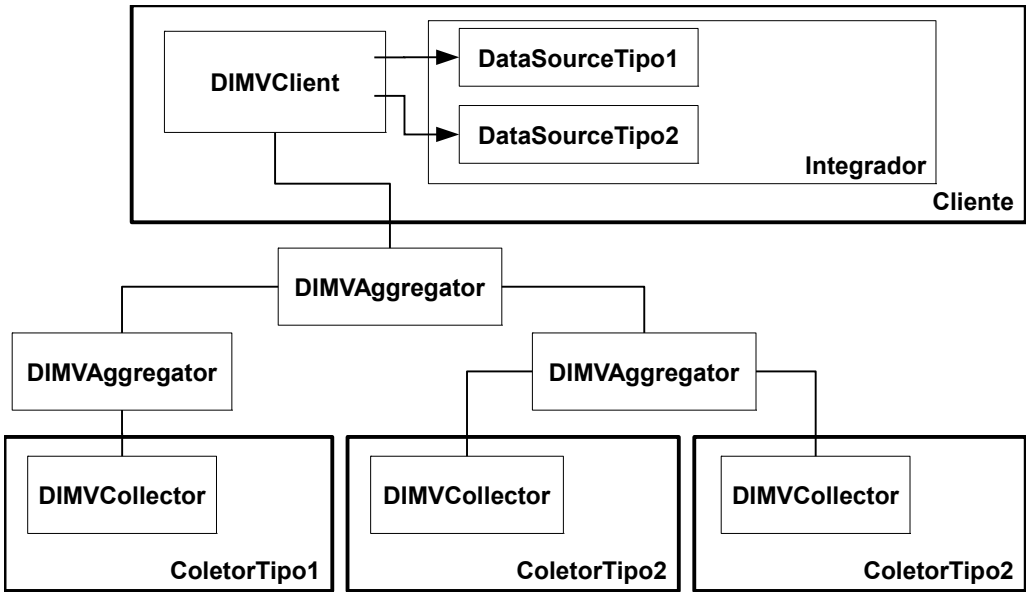


Figura 5.2: Hierarquia do modelo de coleta mostrando as classes implementadas

A classe `DIMVCollector` implementa as funcionalidades do coletor que são independentes da fonte de dados. A inicialização de um objeto dessa classe é mostrada, de maneira simplificada, no fluxograma da figura 5.3. Nela, inicialmente são salvos o identificador e o tipo do coletor, que são recebidos como parâmetros do método de inicialização. Após, é criado o dicionário que servirá para armazenar os subscritores do coletor. Então, é inicializada a *thread de conexão*, que é responsável pela comunicação com objetos remotos. Para ela é passado o terceiro parâmetro recebido, que é um vetor de nomes de agregadores. Essa thread é necessária para que o programa não bloqueie quando estiver sendo executado o `RunLoop`, conforme explicado na introdução desse capítulo. Por fim é retornada um referência ao objeto inicializado.

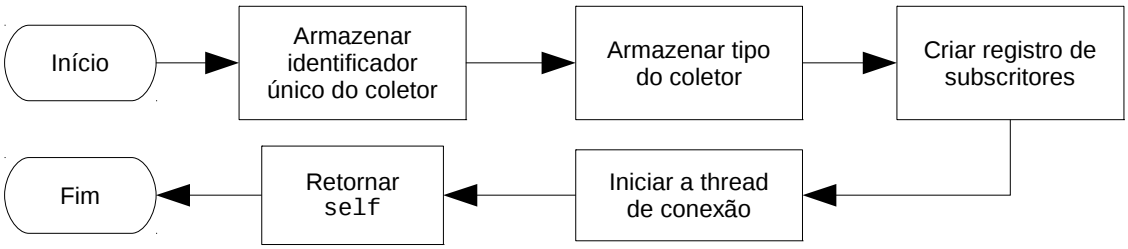


Figura 5.3: Fluxograma simplificado da inicialização de um `DIMVCollector`

A figura 5.4 mostra o funcionamento da thread de conexão do `DIMVCollector`. Nela é executado um laço que itera pelos nomes de agregadores fornecidos na inicialização do coletor. Para cada nome, é obtida uma referência ao objeto remoto associado a

ele no servidor de nomes. Essas referências são armazenadas em um dicionário, usando seus identificadores únicos como chave. Além disso, o coletor registra-se em cada um dos agregadores, através de uma chamada remota. Nesse registro é passada uma referência ao coletor. Os registros de coletores são repassados pelos agregadores para seus componentes superiores. Dessa forma, eles chegam aos clientes conectados na hierarquia. O laço termina quando todos os agregadores, cujos nomes foram recebidos como argumento, houverem sido conectados. Então, é executado o `RunLoop` padrão da thread. Esse é responsável por receber as chamadas remotas e executar métodos que outras threads solicitarem que sejam executados na thread de conexão. Esse é o caso de quando a thread principal quer enviar dados para os agregadores.

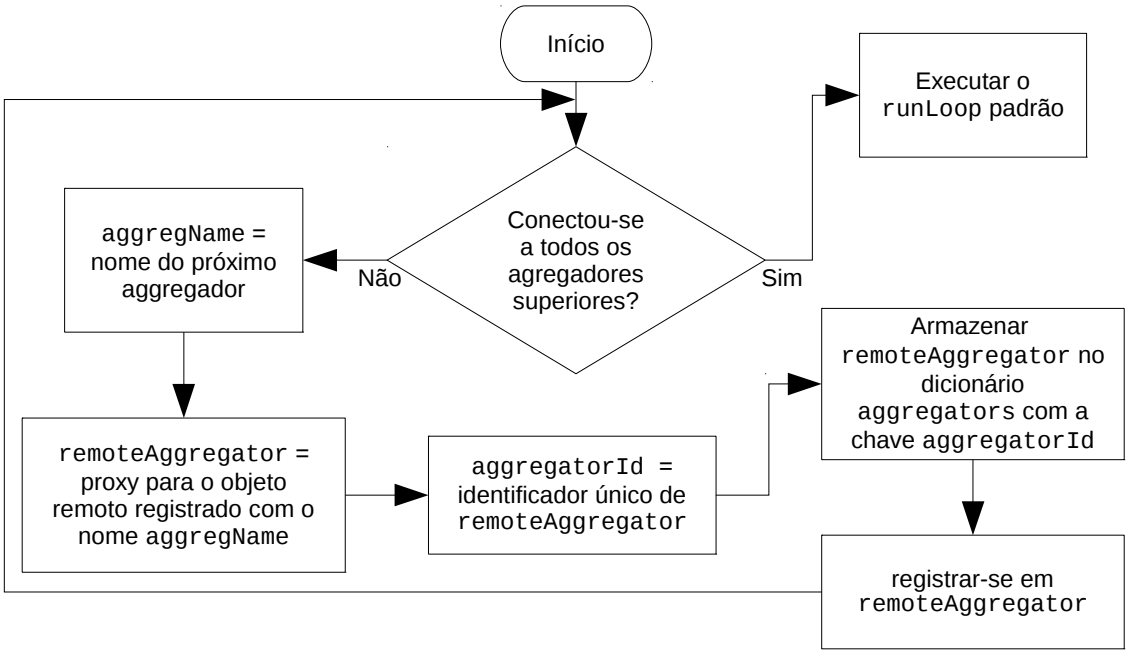


Figura 5.4: Fluxograma simplificado da inicialização da thread de conexão de um `DIMVCollector`

Objetos da classe `DIMVCollector` recebem dados de monitoramento através do método `receiveData:`. Esses dados devem estar na forma de um objeto de uma classe que implemente o protocolo NSCoding, da biblioteca base da linguagem Objective-C. Isso assegura que o objeto pode ser copiado para um objeto remoto. Os dados recebidos são repassados aos agregadores através de uma chamada remota a seu método `receiveData:`, implementado na classe `DIMVAggregator`. O processo de envio de dados pelo coletor, na thread de conexão, é ilustrado na figura 5.5. Isso é feito pelo `RunLoop`, por solicitação da thread principal, que passa como argumento o dado a ser enviado. Primeiro, é obtido um enumerador para o dicionário de agregadores subscritos ao coletor. Então, tem-se um laço que itera pelos objetos enumerados, enviando o dado para cada um deles. Com o fim do laço, dá-se o retorno do método.

Objetos da classe `DIMVAggregator`, ao serem inicializados, registram-se em um servidor de nomes. Dessa maneira eles podem ser encontrados por outros componentes. Assim, possibilita-se conexão desses com os agregadores. Esse processo de inicialização é mostrado, simplificadamente, no fluxograma da figura 5.6. Primeiramente, são armazenados em atributos o nome e o identificador únicos do agregador. Esses são recebidos como parâmetros do método de inicialização. Depois, o objeto é registrado, com o nome

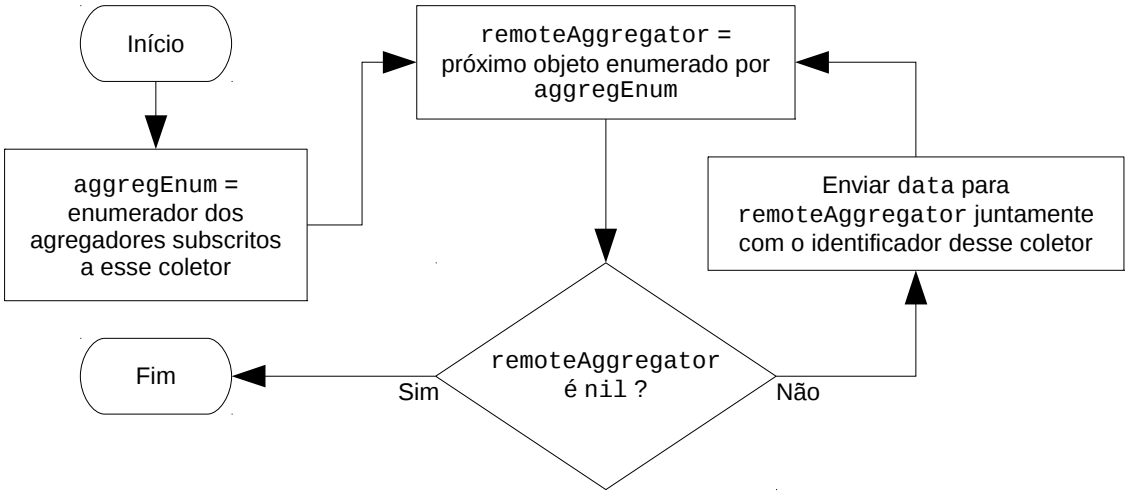


Figura 5.5: Fluxograma simplificado do envio de dados por um `DIMVCollector`

recebido, no servidor de nomes. O terceiro argumento é opcional, constando de um vetor de nomes de agregadores de nível hierárquico superior. Caso não tenha sido recebido esse vetor, então o método termina, com o retorno do objeto inicializado. Caso contrário, inicia-se um laço que itera pelo vetor de nomes de agregadores. Para cada nome, obtém-se uma referência para o objeto remoto associado, utilizando o servidor de nomes. Esse objeto deve ser um agregador de nível hierárquico superior. Então, o agregador registra-se como subordinado desse superior, através de chamada remota. Nessa é passada uma referência a si mesmo, como parâmetro. O próximo passo é repassar ao superior os registros de coletores que o agregador possuir. Isso é necessário porque esses registros podem ter sido recebidos durante a inicialização do agregador, pois subordinados não sabem que o agregador ainda não foi completamente inicializado. É interessante lembrar que, após inicializados, os componentes recebem essa informação durante o processo de registro de cada coletor. Após a saída do laço, a inicialização termina através do retorno de uma referência para o objeto inicializado.

O recebimento de dados de monitoramento por um `DIMVAggregator` é feito quando algum objeto remoto chama seu método `receiveData:`. Os agregadores repassam os dados recebidos através de chamadas remotas aos métodos `receiveData:` dos objetos de destino. Esses podem ser tanto da classe `DIMVClient` quanto da `DIMVAggregator`. Nesse repasse é enviado também o identificador único do coletor que fez o envio inicial das informações. A figura 5.7 mostra um fluxograma simplificado desse processo. Inicialmente é obtido um enumerador do dicionário de componentes registrados para receber dados do coletor cujo identificador é recebido como argumento (subscritores). Então vai-se para um laço que itera entre os objetos enumerados por ele. A cada iteração, os dados são enviados para um desses objetos, juntamente com o identificador do coletor. O método termina com a saída do laço.

Na sua inicialização, um objeto da classe `DIMVClient` recebe um identificador e uma lista de nomes de agregadores a conectar-se. Esse processo é ilustrado pelo fluxograma da figura 5.8. O primeiro passo mostrado é o armazenamento do identificador do cliente. Após, é iniciada uma *thread de conexão*. Para essa inicialização é passado como argumento o vetor de nomes de agregadores citado anteriormente. Após, finaliza-se a inicialização, retornando uma referência ao objeto inicializado.

O funcionamento da thread de conexão do `DIMVClient` é mostrado na figura 5.9,

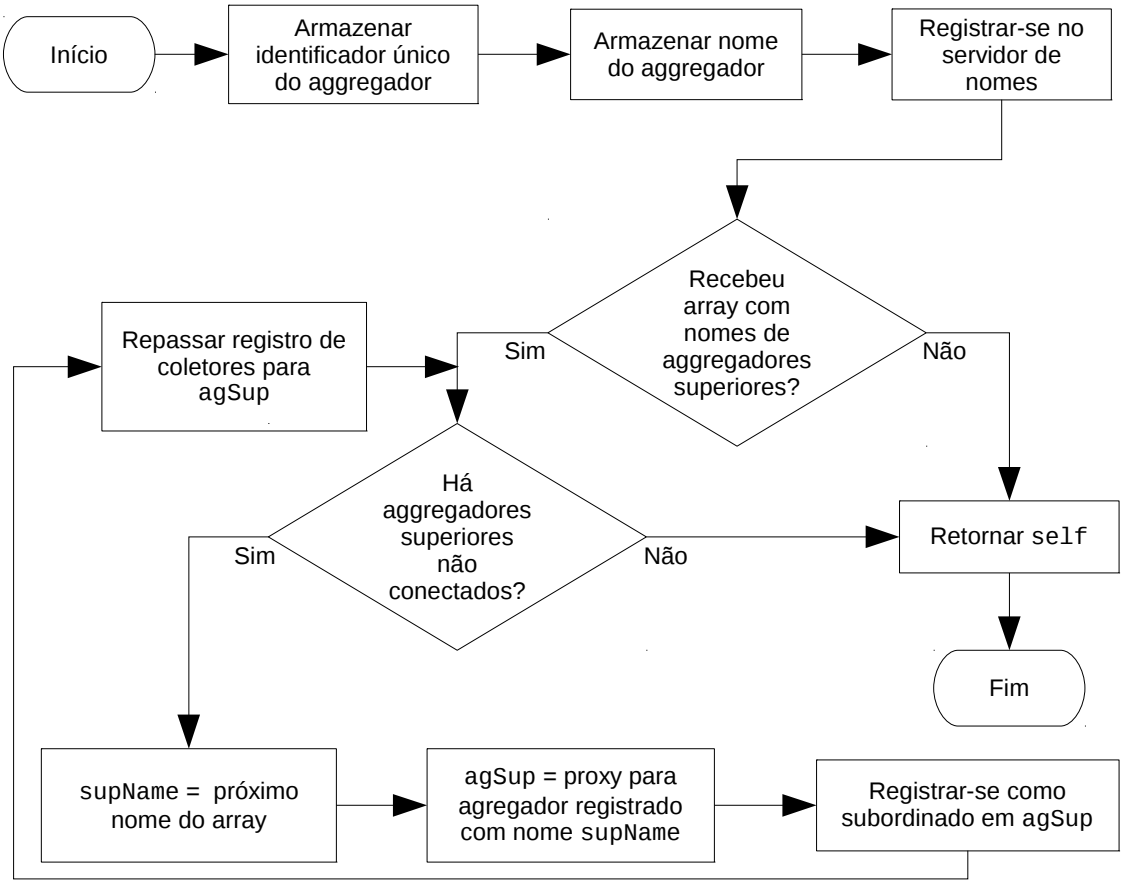


Figura 5.6: Fluxograma simplificado da inicialização de um `DIMVAggregator`

em um fluxograma. Ele consta de um laço que itera pelo vetor de nomes de agregadores a serem conectados. Esse parâmetro é passado pelo método de inicialização do `DIMVClient`. Para cada um desses nomes, é obtido um proxy para o objeto correspondente, utilizando o servidor de nomes. Esse é uma referência a um objeto remoto da classe `DIMVAggregator`. Então o `DIMVClient` registra-se no agregador, passando como argumento uma referência para si mesmo. Após, ele solicita ao agregador os registros de coletores que ele possui. Cada um desses registros armazena o identificador de sua origem, que é alterado pelo cliente, para associá-lo ao agregador que os forneceu. Esse campo vai ser utilizado para o encaminhamento de solicitações de subscrição. Então, esses registros são salvos no cliente. Os registros subsequentes serão repassados ao cliente pela hierarquia de agregadores, conforme explicado anteriormente. Quando não houver mais agregadores a serem conectados, o laço termina e é executado o `RunLoop` padrão da thread. Dessa forma a thread fica pronta para responder a chamadas remotas e execução de métodos por solicitação de outras threads.

Assim como a `DIMVAggregator`, a classe `DIMVClient` recebe dados como argumento de seu método `receiveData:`. O cliente utiliza o tipo do coletor para descobrir a que objeto deve repassar os dados recebidos. Então ele chama o método `inputData:` desse objeto, passando como argumento os dados recebidos. O uso que será feito dos dados pelo receptor não interessa ao DIMVHCM. Esse processo é ilustrado, simplificadamente, no fluxograma da figura 5.10. Inicialmente, é mostrada a obtenção do registro do coletor que originou o dado. Para isso utiliza-se o identificador do coletor, que é um dos argumentos do método de recebimento de dados. O outro argumento é o dado propri-

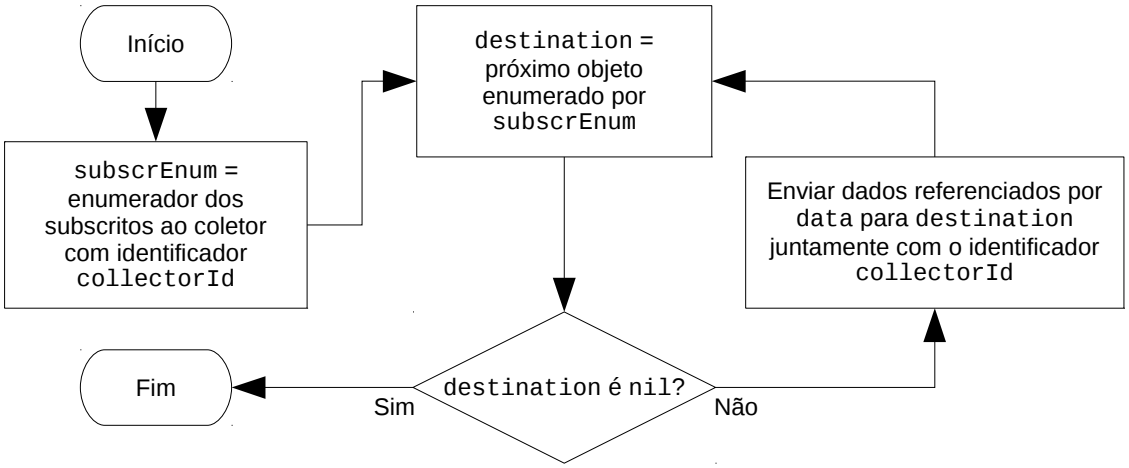


Figura 5.7: Fluxograma simplificado do recebimento de dados por um `DIMVAggregator`

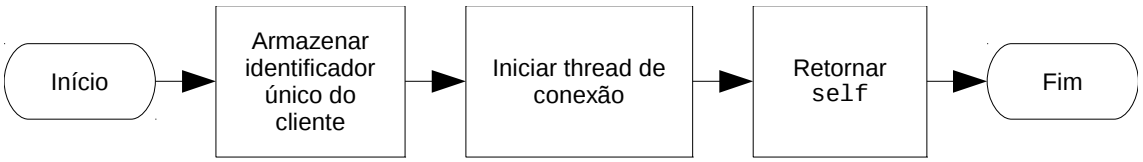


Figura 5.8: Fluxograma simplificado da inicialização de um `DIMVClient`

amente dito. A seguir, é obtido o tipo do coletor, que é uma das informações armazenadas em seu registro. Utilizando esse tipo como chave, é buscado o objeto processador de dados associado a ele. Então, os dados de monitoramento são repassados ao processador de dados e finaliza-se o método.

Utilizando as três classes previamente explicadas, implementou-se coletores, um agregador e um cliente. O cliente usa uma versão modificada do protótipo do DIMVisual que usa um `DIMVClient` para receber os dados. Essa versão do DIMVisual foi utilizada então para construir um módulo de leitura para o TRIVA. Com isso, a ferramenta pode obter dados integrados, no formato Pajé. O resultado final é uma visualização gerada a partir dos dados fornecidos pelo DIMVisual.

A subseção a seguir aborda o funcionamento do mecanismo de subscrição implementado no protótipo do DIMVHCM. Logo após, em uma nova seção, são apresentados detalhes da implementação do cliente citado no parágrafo anterior.

5.1.1 Subscrição

Nessa subseção são mostrados detalhes do processo de subscrição a um coletor. Em especial, é destacado o processamento de solicitações de subscrição por um agregador, um cliente e um coletor.

O processo realizado quando um agregador recebe uma solicitação de subscrição é ilustrada na figura 5.11, de forma simplificada. Essa contém um fluxograma em que o primeiro passo é a obtenção de um registro de coletor, utilizando seu identificador. Esse é fornecido como argumento ao método que processa a solicitação. O segundo argumento é o identificador do componente que realizou a solicitação. O agregador procura pela referência ao objeto remoto correspondente. Isso é feito primeiro nos registros de agregadores e, caso não encontrado, no registro de clientes. Após, o componente encontrado

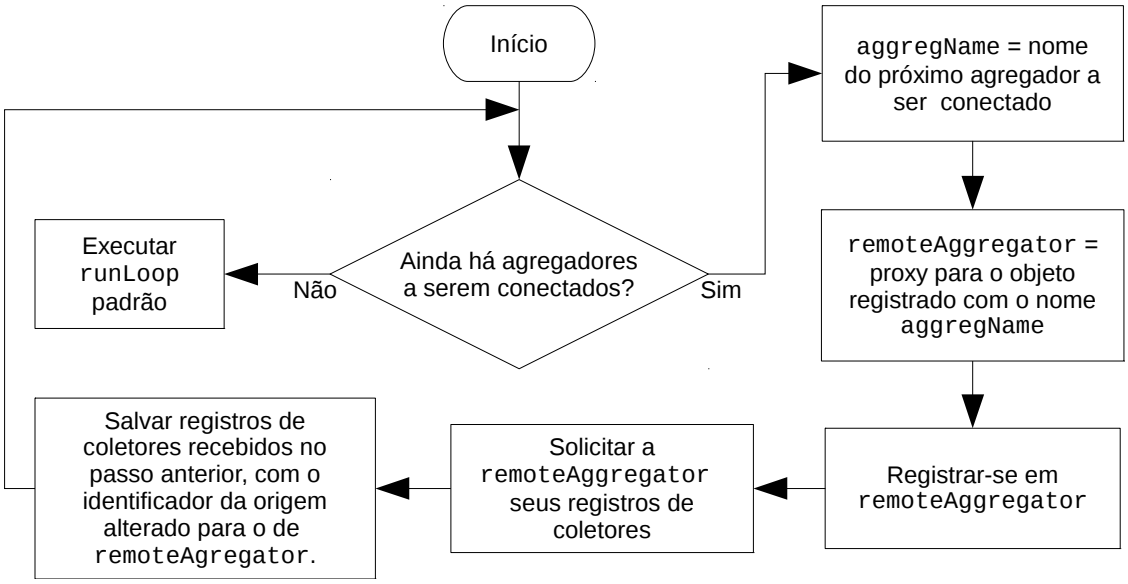


Figura 5.9: Fluxograma simplificado da inicialização da thread de conexão de um `DIMVClient`

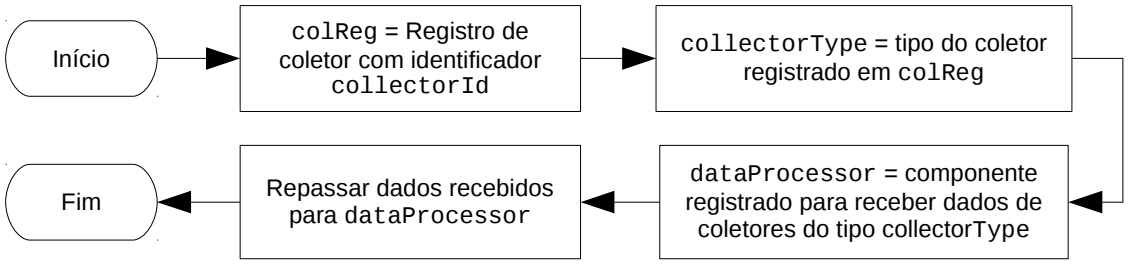


Figura 5.10: Fluxograma simplificado do recebimento de dados por um `DIMVClient`

é armazenado no dicionário de subscritores do coletor. Esse dicionário faz parte do registro do coletor, obtido anteriormente. Após, é verificado se o agregador já possui uma subscrição ao coletor a que foi solicitada a subscrição. Caso positivo, o método termina. Caso negativo, é necessário que o agregador subscreva-se para receber dados do coletor. Para isso, primeiro é obtido o identificador da origem do coletor, que está em seu registro. Essa origem é o componente que repassou o registro ao agregador. Usando o identificador obtido, é buscada uma referência ao objeto remoto. Primeiramente, essa busca é feita entre os agregadores registrados. Caso não seja encontrado, ele é buscado nos registros de coletores. Após, é adicionado o identificador do coletor na lista de coletores para os quais o agregador possui subscrição. Por último, é realizada uma chamada remota ao objeto de origem do coletor, solicitando a subscrição do agregador ao coletor.

A classe `DIMVClient` implementa métodos que podem ser chamados por um objeto que vai fazer o gerenciamento das subscrições. Esse objeto deve implementar um protocolo chamado `HCMSubscriptionHandler`. Esse define métodos que serão chamados pelo cliente quando um novo coletor for registrado ou derregistrado. O gerenciador, então, pode realizar a ação adequada a cada um desses casos. O `DIMVClient` implementa métodos para especificar um gerenciador e para que esse subscreva-se e cancele subscrições a coletores.

Na figura 5.12 é mostrado um fluxograma do processamento de uma requisição de subscrição por um objeto da classe `DIMVClient`. Essa requisição é feita por um geren-

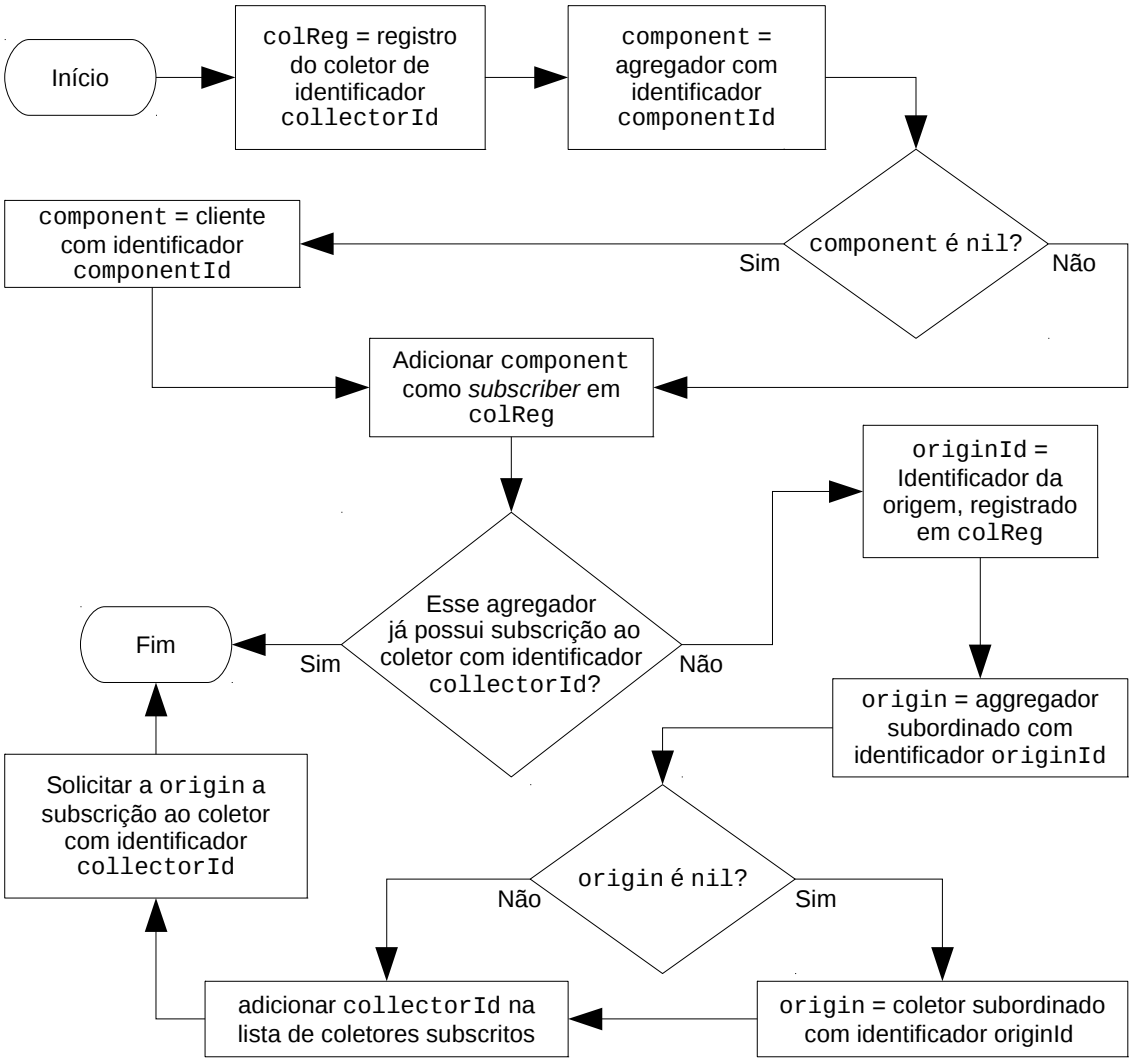


Figura 5.11: Fluxograma simplificado do processamento de uma solicitação de subscrição por um `DIMVAggregator`

ciador de subscrição, conforme explicado no parágrafo anterior. Olhando a figura, vê-se que o primeiro passo é a obtenção do registro do coletor do qual vai ser feita a subscrição para receber dados. Então, desse registro obtêm-se o identificador da origem. No passo seguinte, ele é usado para obter uma referência ao objeto remoto de origem, do dicionário de agregadores conectados. Feito isso, é realizada uma chamada remota a esse objeto, solicitando a subscrição do cliente ao coletor. Essa chamada é realizada pela thread de conexão, conforme ilustra a figura 5.13. Os parâmetros dessa chamada são os identificadores únicos do cliente e do coletor.

No protótipo, por padrão, o DIMVisual é o gerenciador de subscrição e subscreve-se a todos os coletores disponíveis. Foi implementado um método na biblioteca do DIMVisual que permite modificar o gerenciador de subscrição. Esse método é usado pelo TRIVA para especificar um gerenciador que fornece uma interface gráfica através da qual o usuário pode selecionar os coletores aos quais subscrever-se. Esse gerenciador está em vias de ser portado para a última versão do TRIVA que utiliza um diferente biblioteca gráfica em relação à versão em que o gerenciador de subscrições foi implementado.

O processamento de uma requisição de subscrição por um coletor é mostrado, simplificadamente, na figura 5.14. Primeiro ele obtém uma referência ao agregador que fez a

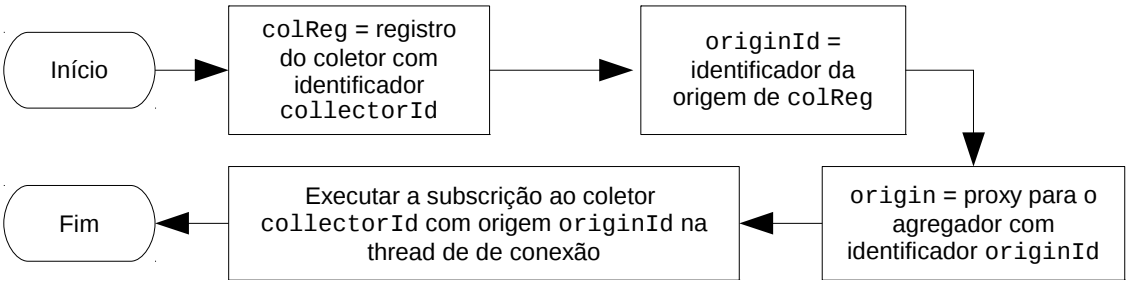


Figura 5.12: Fluxograma simplificado do processamento de uma solicitação de subscrição por um `DIMVClient`

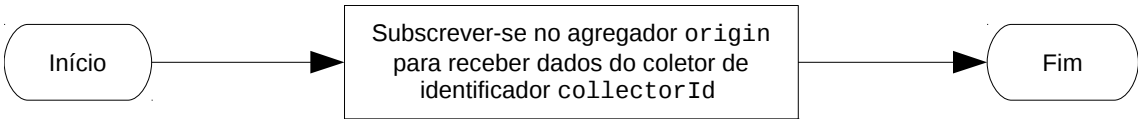


Figura 5.13: Fluxograma simplificado do envio de uma solicitação de subscrição na thread de conexão de um `DIMVClient`

solicitação, a partir de seu dicionário de agregadores conectados. Então, ele adiciona essa referência no dicionário de agregadores subscritos.

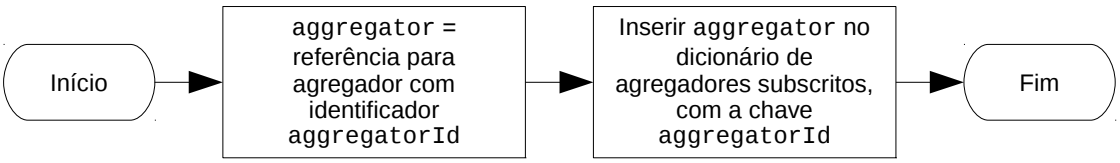


Figura 5.14: Fluxograma simplificado do envio de uma solicitação de subscrição por um `DIMVCollector`

5.2 Implementação do Cliente

Um cliente do DIMVHCM foi implementado integrando um `DIMVClient`, com os protótipos do DIMVisual e TRIVA. A figura 5.15 representa a arquitetura desse cliente. O `DIMVClient` fornece dados aos *data sources* do DIMVisual. Desses, os que possuem dados armazenados (não estão vazios) ficam em uma fila dentro de um componente do DIMVisual, chamado *Order*. O componente principal do DIMVisual é chamado *integrador* e é responsável por obter os dados convertidos do *Order*. Um módulo do TRIVA chamado `DIMVHCMReader`, possui um produtor que utiliza um integrador do DIMVisual para receber dados e os coloca em uma fila. Ainda no *reader*, um *consumidor* retira os dados da fila e os fornece ao componente seguinte do TRIVA, que gera a visualização.

O restante dessa seção trata com mais detalhes as modificações e adições feitas na ferramentas utilizadas para implementar o cliente. A subseção a seguir trata das modificações realizadas no DIMVisual. Na subseção 5.2.2 fala-se sobre as modificações feitas no TRIVA, incluindo a implementação do `DIMVHCMReader`.

5.2.1 Integração ao DIMVisual

No protótipo implementado foi necessário modificar o DIMVisual para receber dados de um `DIMVClient` e para suportar o monitoramento *on-line*. As maiores modificações

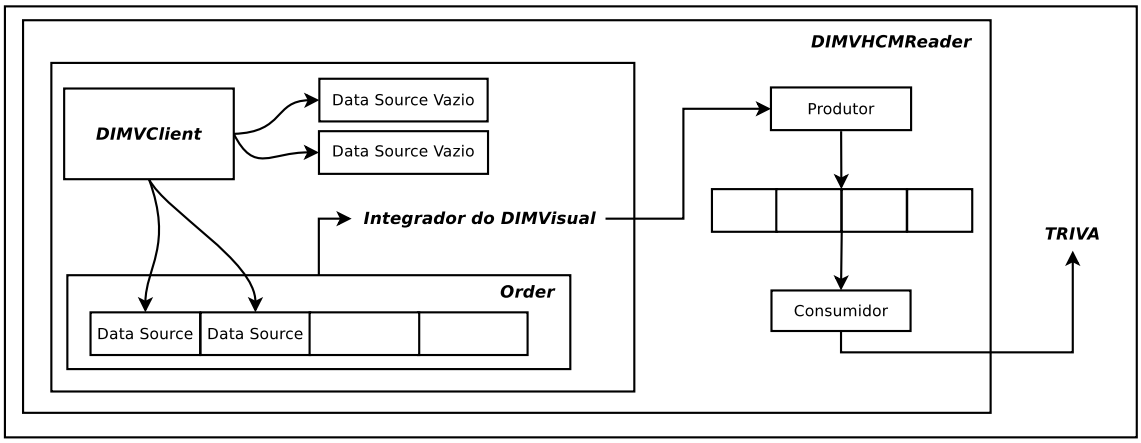


Figura 5.15: Arquitetura do Cliente Implementado

tiveram a ver com os *data sources* e com o funcionamento do componente *Order*. Esse último possui uma fila de prioridade em que são armazenados data sources. Ela é utilizada para buscar o *data source* que possui o dado mais antigo.

A cada tipo de coletor deve ser associado um *data source* do DIMVisual. Nesses deve estar implementado o método `inputData`, que é chamado pelo `DIMVClient` para repassar os dados recebidos. Então o `DIMVClient` foi configurado para encaminhar os dados dos coletores para os *data sources* apropriados.

A implementação do componente *Order* estava feita de maneira que *data sources* eram retirados da fila quando não possuíam mais dados. Quando a fila tornava-se vazia, o programa terminava. Agora, no entanto, como estamos fazendo monitoramento *on-line*, um *data source* pode ficar temporariamente sem dados. Caso em que seria retirado da fila. Por isso foi implementado um mecanismo para que esses possam adicionar-se na fila quando estiverem fora da mesma (vazios) e receberem novos dados. Essa inserção pode ser vista no fluxograma da figura 5.16, que mostra de forma simplificada o processo de recebimento de dados de monitoramento por uma fonte de dados do DIMVisual. O primeiro passo mostrado é a geração de eventos genéricos através do processamento dos dados recebidos. Após, esses eventos são armazenados internamente. Caso o data source não possuísse eventos armazenados antes dessa inserção, então ele chama um método do componente *Order* para solicitar que seja colocado na fila dele. Com isso, o procedimento de recebimento dos dados é finalizado. O mecanismo de inserção de *data sources* no *Order* em tempo de execução foi uma das adições realizadas nessa nova versão.

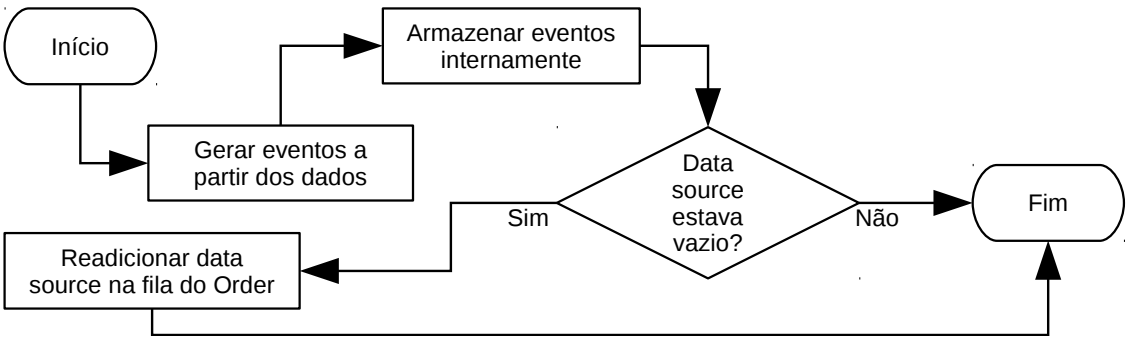


Figura 5.16: Fluxograma simplificado do recebimento de dados por uma fonte de dados do DIMVisual

Também foi feito com que o programa não termine quando a fila do *Order* se esvazia. Ao invés disso, o integrador fica bloqueado, esperando por novos dados. O fluxograma da figura 5.17 mostra, simplificadamente, o funcionamento do método em que o componente *Order* fornece dados convertidos. Inicialmente o objeto fica bloqueado até que haja algum *data source* na fila. Isso é feito através do uso de um objeto da classe `NSConditionLock`, que implementa o bloqueio condicional, para prover acesso exclusivo à fila. Ao solicitar o bloqueio a esse objeto, deve-se fornecer uma condição. A thread então ficará bloqueada até que a condição seja satisfeita e o *lock* não pertença a outra thread. Quando um data source é colocado na fila, ela modifica a condição para indicar que não está vazia. Quando a fila torna-se vazia, é modificada a condição para a que indica esse estado. No próximo passo, é retirada uma fonte de dados da fila de prioridade. Na verdade, os dois primeiros passos consistem de uma única chamada a um método da fila. A seguir, é chamado um método da fonte de dados, que fornece um evento convertido. Caso haja mais eventos no data source, ele será recolocado na fila de prioridade. A data fornecida pelo data source é utilizada para ordenar a fila. Essa é a data do evento mais velho armazenado por ele, que é modificada quando da retirada de um evento. Por isso é feita essa retirada e recolocação na fila de prioridade. O próximo passo mostrado no fluxograma é o retorno do evento convertido, finalizando a execução do método.

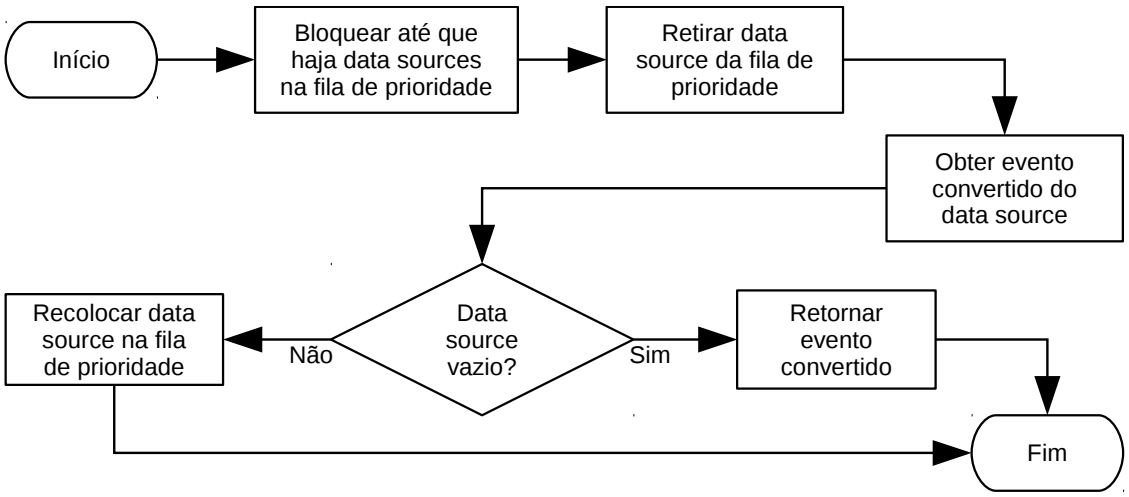


Figura 5.17: Fluxograma simplificado do fornecimento de dados convertidos pelo componente `Order` do DIMVisual

O DIMVisual pode ser usado tanto como um programa quanto como uma biblioteca, dentro de outro programa. As modificações aqui descritas foram feitas para ambas dessas formas. Também foi necessário implementar métodos para receber os argumentos para a configuração do `DIMVClient`. No caso de uso como programa, isso é feito por argumentos de linha de comando. Para o uso como biblioteca, foi implementado um método que recebe esses argumentos.

5.2.2 Integração com o TRIVA

O TRIVA foi utilizado no protótipo do DIMVHCM, para gerar visualizações dos dados coletados. Uma visão geral dessa ferramenta foi apresentada na seção 3.2. Para esse trabalho foi utilizada a abordagem de visualização com *treemap*.

A integração do TRIVA com o DIMVHCM foi feita através da implementação de um módulo de leitura que obtém dados do DIMVisual. Além disso, foi adicionado ao

TRIVA uma interface para controle de subscrições. Através dessa, é possível selecionar de quais coletores o cliente deve receber dados. Adicionalmente, o TRIVA já possibilita a filtragem local dos dados a serem mostrados na visualização. Dessa forma é possível escolher especificamente quais medidas serão mostradas.

5.2.2.1 DIMVHCM Reader

A versão previamente existente do protótipo do TRIVA, realizava apenas visualização *post-mortem* dos dados de monitoramento. Mais especificamente, as informações a serem visualizadas eram lidas de arquivos. Esses deveriam conter todas as informações a serem visualizadas. Por isso foi necessário desenvolver um módulo leitor para o TRIVA receber dados do DIMVisual, conforme publicado em um trabalho anterior (TESSER; SCHNORR; NAVAUX, 2009). Além disso, foi necessário implementar aspectos relacionados à atualização da visualização, em resposta à chegada de novos dados. Dessa forma, obteve-se a capacidade de visualização *on-line* das informações de monitoramento.

O módulo desenvolvido é chamado `DIMVHCMReader`. Esse substitui o módulo de leitura de arquivo, conforme ilustrado na figura 5.18. Ele utiliza o modelo produtor-consumidor. A thread produtora faz requisições de eventos ao integrador do DIMVisual, também mostrado na figura. Esse responde com eventos no formato Pajé, que são armazenados pelo produtor em uma fila. Se o DIMVisual não possuir eventos quando do recebimento de uma requisição desses, ele bloqueia, até que cheguem novos dados de monitoramento. Ocorrendo isso, o produtor também fica bloqueado. O consumidor executa em outra thread. Ele é responsável por retirar os dados da fila abastecida pelo produtor e fornecê-los ao componente seguinte do pipeline de processamento do TRIVA.

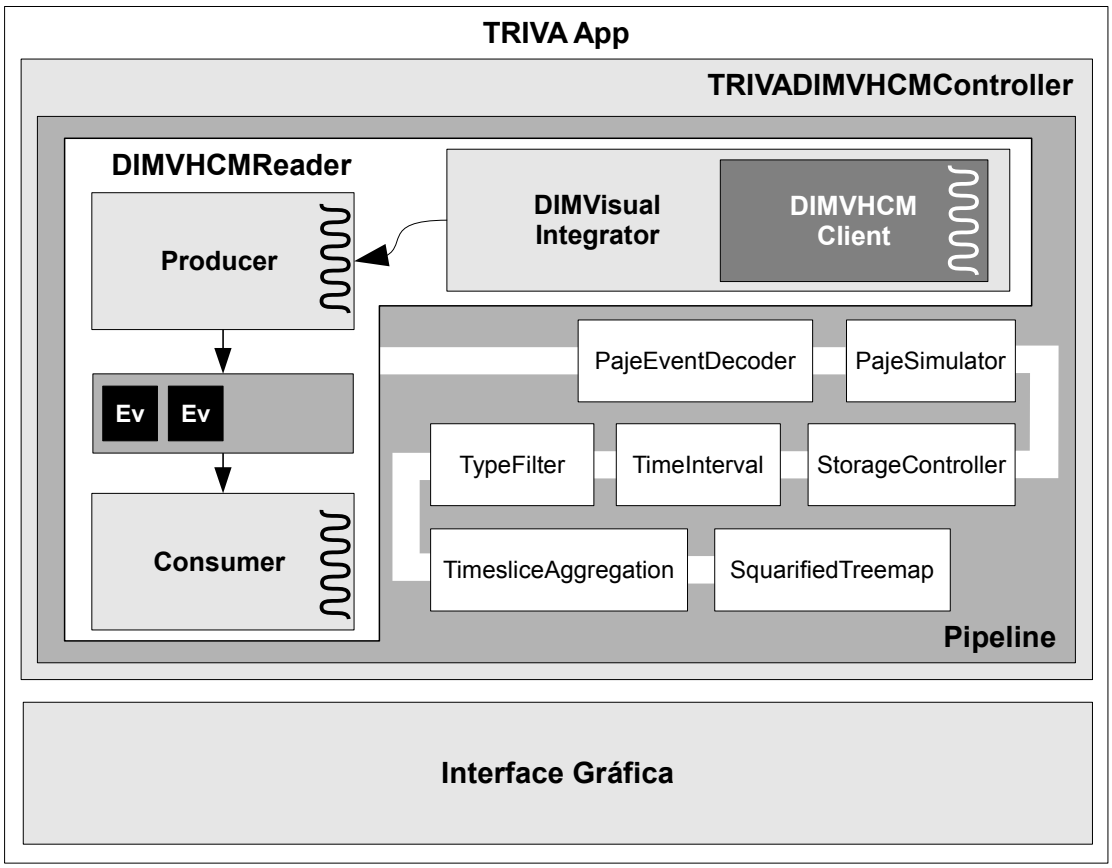


Figura 5.18: Diagrama de componentes do TRIVA com o DIMVHCMReader

Ambas threads, produtora e consumidora, são iniciadas por um componente chamado `TRIVADIMVHCMController`. Esse também é responsável pela inicialização do pipeline do TRIVA. A inicialização desse componente é realizada pela função principal (`main`) do TRIVA, que também é responsável por iniciar a interface gráfica. A thread produtora executa um método do `DIMVHCMReader`. A consumidora executa um implementado no `TRIVADIMVHCMController`, utilizando métodos do *reader*.

A figura 5.19 contém um fluxograma mostrando, de forma simplificada, o processo de inicialização do `TRIVADIMVHCMController`. Primeiro, é inicializado o pipeline do TRIVA. Isso compreende a inicialização dos componentes e a atribuição de seus componentes de entrada e saída. Ou seja, dos objetos de quem receberão e para quem enviarão os dados. A seguir, referências a alguns dos componentes são armazenadas em atributos, para uso futuro. O próximo passo é a leitura de um arquivo de configuração cujo nome é passado através da linha de comando. Esse contém os nomes dos *data sources* a serem carregados, o nome e identificador únicos do cliente do DIMVHCM e os nomes dos agregadores aos quais esse deve conectar-se. Após sua leitura, essas configurações são passadas para o `DIMVHCMReader`. O método que recebe essas informações é também responsável pela inicialização dos *data sources* do DIMVisual. Após, é configurado o tamanho do *chunk*. *Chunks* são pedaços em que os dados são divididos para serem repassados entre os componentes do pipeline. A seguir, inicializam-se as threads produtora e consumidora, cujo funcionamento será explicado a seguir. Feito isso, o método termina com o retorno de uma referência para o objeto inicializado.

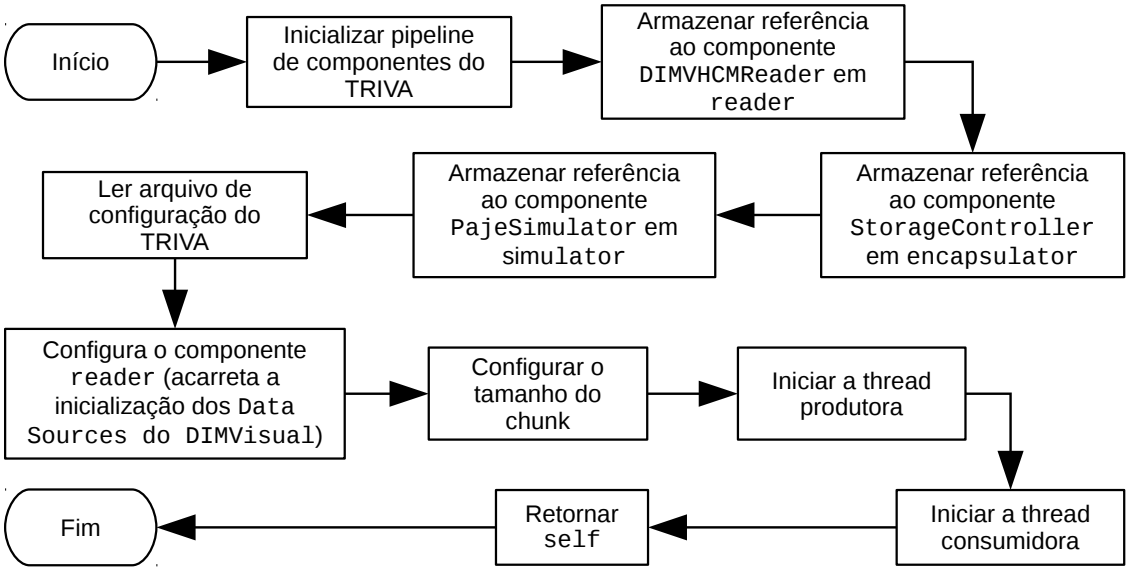


Figura 5.19: Fluxograma simplificado da inicialização do `TRIVADIMVHCMController`

Na figura 5.20 temos uma simplificação do funcionamento da thread produtora do `DIMVHCMReader`. O primeiro passo é a inicialização do `DIMVClient` através de uma chamada a um método do integrador do DIMVisual. Nessa são passados argumentos recebidos do `TRIVADIMVHCMController`, para configuração do cliente. Após, são obtidos do DIMVisual o cabeçalho dos dados, conforme o formato de entrada do Pajé, e esses são colocados em um buffer. Após, entra-se em um laço no qual a cada iteração solicita-se um evento convertido ao integrador do DIMVisual e coloca-se ele no buffer. Caso o DIMVisual retorne `nil`, então o laço termina, assim como o produtor.

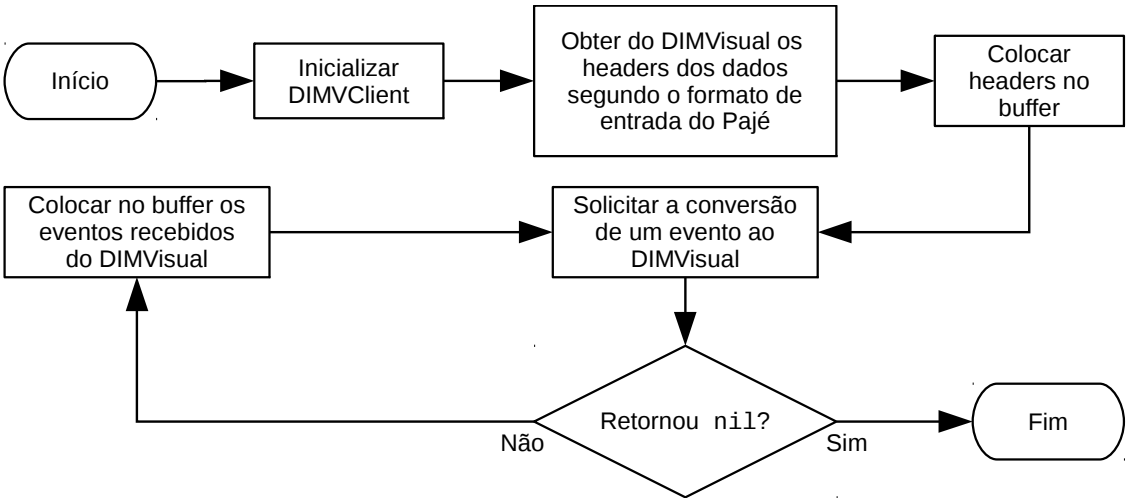


Figura 5.20: Fluxograma simplificado da thread produtora do `DIMVHCMReader`

O fluxograma da figura 5.21, mostra o funcionamento da thread consumidora. A execução consiste em um laço em que a cada iteração o `DIMVHCMReader` lê um novo chunk e esse é repassado para o componente seguinte no pipeline do TRIVA. Primeiramente obtém-se o número do próximo chunk, que é igual a contagem dos elementos do vetor `chunkDates`. Esse valor é armazenado na variável `chunkNumber`. Então, é chamado um método do *reader* que solicita que o chunk de número `chunkNumber` seja iniciado. Isso iniciará um novo chunk, devido ao número passado ser maior que a quantia de chunks lidos. Após, é inserida a data atual no vetor `chunkDates`. A seguir, é chamado o método do *reader* que faz a leitura do chunk. Feito isso, é chamado o método que diz para o *reader* finalizar a leitura do chunk. Esse método recebe como argumento um valor booleano que diz se esse chunk é o último que deve ser lido. No caso do protótipo, o chunk nunca será o último. O próximo passo é a chamada a um método do objeto referenciado por `encapsulator`, para comunicar a mudança nos limites de tempo. O último passo do laço é a execução do `runLoop` padrão da thread, através de uma chamada que permite estipular um limite de tempo para sua execução. Isso é necessário devido ao `encapsulator` utilizar chamadas que precisam ser executadas pelo `runLoop`. Na versão original do TRIVA, como havia apenas uma thread, essa chamada não era necessária. Nesse caso, era utilizado um `RunLoop` que é executado pelas API's gráficas do GNUstep.

A figura 5.22 mostra o que acontece quando é chamado o método `readChunk` do `DIMVHCMReader`. Esse é método é chamado pela thread consumidora. O único parâmetro recebido é o número do chunk a ser lido. Caso o chunk já tenha sido lido, então ele é relido do arquivo de saída e fornecido ao próximo componente do pipeline. Feito isso, o método retorna. Caso contrário entra-se em um laço. Dentro desse, primeiro retira-se um evento do buffer do `DIMVHCMReader`. Após, é chamado um método que testa se o chunk pode ser terminado antes desse evento e, caso negativo, fornece o evento ao componente de saída do *reader*. O resultado do teste é retornado pelo método. Caso o retorno seja negativo, então repete-se o laço, para ler o próximo evento. Caso seja positivo, o evento é colocado de volta na primeira posição do buffer, para que seja retirado na próxima vez que esse método seja chamado. O passo seguinte é a gravação do chunk no arquivo de saída, após o qual o método retorna.

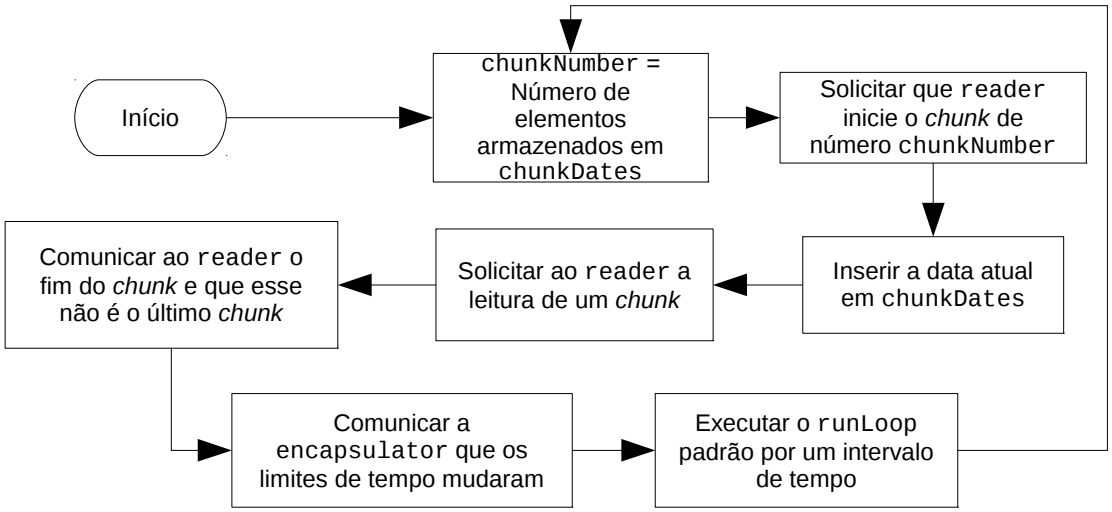


Figura 5.21: Fluxograma simplificado da thread consumidora de dados no `TRIVADIMVHCMController`

5.3 Considerações Finais

Para avaliar o DIMVHCM, foi implementado um protótipo de ferramenta de monitoramento *on-line*. Nesse, os dados coletados são fornecidos a um cliente que os repassa a fontes de dados do DIMVisual. Esse componente fornece os dados convertidos ao TRIVA, que gera uma visualização on-line das informações.

Nesse capítulo, inicialmente, foram apresentadas funcionalidades da linguagem de programação Objective-C e do framework GNUstep. Esses foram utilizados na implementação tanto do DIMVHCM quanto do DIMVisual e do TRIVA. Foram abordados aspectos relacionados ao uso de objetos remotos nessa linguagem.

Para cada componente do modelo de coleta foi implementada uma classe que realiza as funcionalidades que são independentes do formato dos dados. Essas classes podem ser usadas para implementar componentes específicos para cada fonte de dados. Foram apresentados em maiores detalhes os processos de inicialização e recebimento de dados, implementados nessas classes. Além disso foi mostrado como cada uma delas processa as requisições de subscrição.

O DIMVisual e o DIMVHCM foram utilizados para implementar um cliente. Foi vista a sua arquitetura, mostrando o caminho que os dados percorrem dentro dele. Além disso, foram especificadas modificações realizadas no DIMVHCM e no TRIVA, para que esses suportassem a atualização *on-line*. Em especial, foi apresentado o `DIMVHCMReader`, que é um módulo para o TRIVA, que faz a leitura dos dados provenientes do DIMVisual.

Através de fluxogramas, foi mostrado o processo de recebimento de dados por um *data source* do DIMVisual. Da mesma forma, foi visto o funcionamento do método através do qual o componente `Order` do DIMVisual fornece eventos convertidos para o formato Pajé. Além disso, foi mostrado o funcionamento das threads produtora e consumidora do cliente, incluindo a leitura de um *chunk* pelo `DIMVHCMReader`.

No capítulo seguinte são apresentados experimentos realizados com o protótipo e a avaliação de seus resultados. Foram realizadas medições do tempo de envio de mensagens com diferentes hierarquias e configurações dos coletores. Também foi testada a capacidade do cliente em processar os dados recebidos.

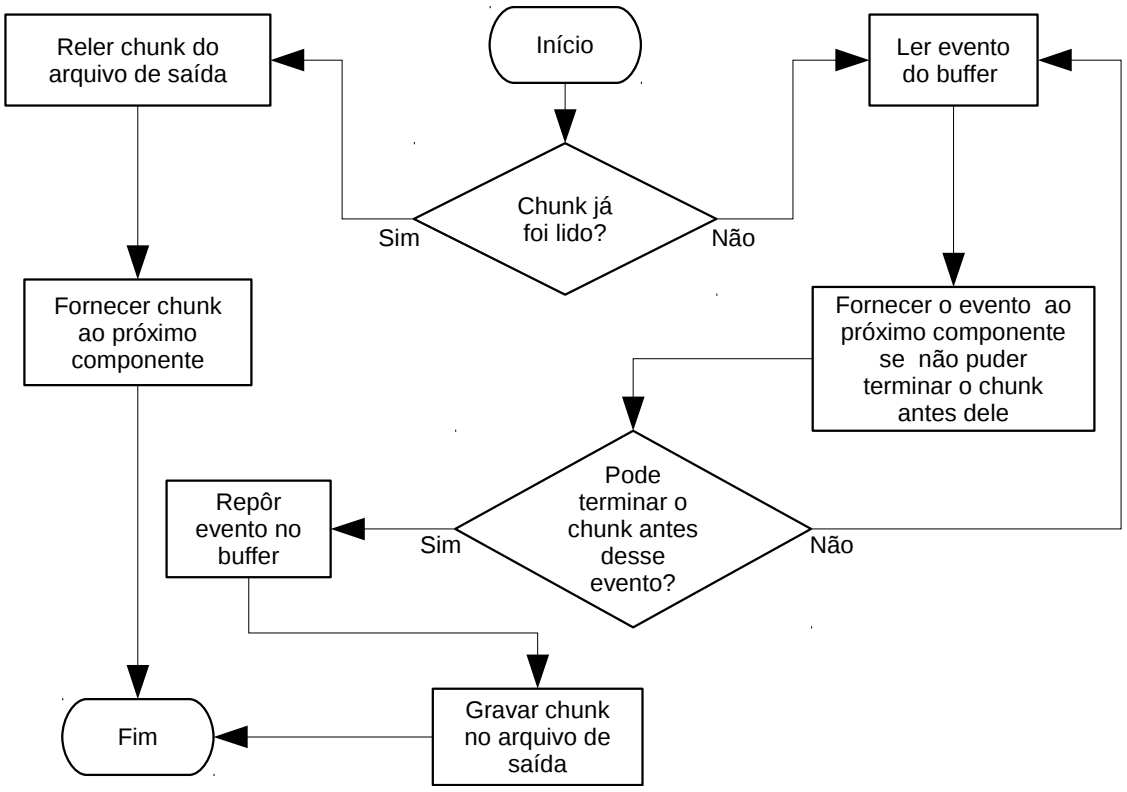


Figura 5.22: Fluxograma simplificado da leitura de um *chunk* pelo `DIMVHCMReader`

6 TESTES E AVALIAÇÃO

Neste capítulo são apresentados experimentos realizados para avaliar o DIMVHCM. Para isso foi utilizado o protótipo apresentado no capítulo 5. Primeiramente são apresentadas medições do tempo necessário para transmitir dados de monitoramento através da hierarquia. Logo após, é apresentada uma avaliação do desempenho do cliente.

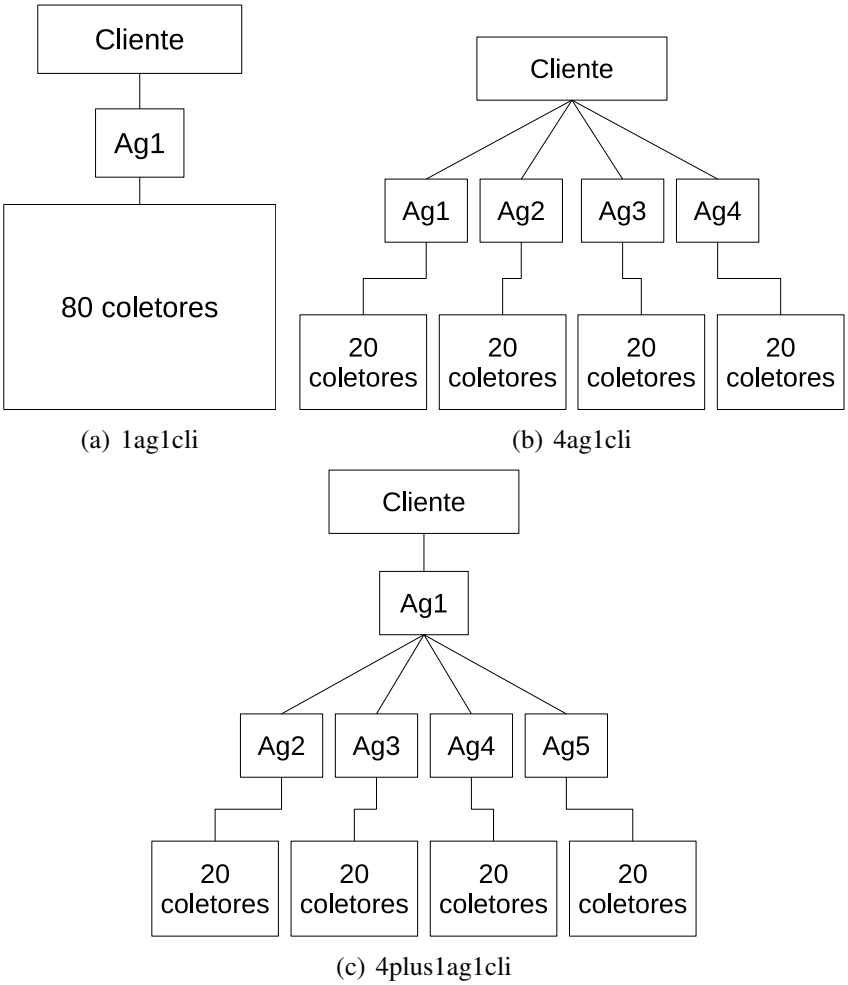
Nesses experimentos, foi utilizada uma implementação de coletor que obtém dados de um `gmond` do Ganglia. Esse foi configurado para não transmitir os dados por multicast e não foi utilizado nenhum `gmetad`. O coletor conecta-se ao daemon do Ganglia, utilizando *sockets*, e obtém dados no formato XML. Esses dados são transmitidos, sem modificação, através da hierarquia do DIMVHCM. Os `gmond` foram configurados para monitorar 9 medidas. O tamanho de cada documento XML obtido do Ganglia, com a configuração utilizada, é de aproximadamente 7 quilobytes.

6.1 Duração da transmissão de dados de monitoramento

Nesse experimento mediu-se quanto tempo os dados de monitoramento, enviados por um coletor, demoram para chegar até um cliente. Para isso, os componentes do protótipo foram instrumentados para registrar o tempo no recebimento e envio dos dados.

Foram realizadas medições utilizando de 1 a 4 coletores por nó. Além desses, os agregadores e clientes foram executados cada um em um nó diferente do mesmo cluster. Nesses testes foram utilizadas as três configurações de hierarquia ilustradas na figura 6.1. Essa figura mostra como ficaria a distribuição dos coletores com 1 coletor por nó. Na hierarquia nomeada *1ag1cli*, tem-se todos os coletores enviando dados para um único agregador, que os retransmite diretamente para um cliente. Na segunda configuração (*4ag1cli*), os nós monitorados são divididos igualmente em quatro grupos. Cada um desses grupos envia dados para um agregador diferente. Todos esses agregadores repassam os dados para o cliente no topo da hierarquia. Por último, foi adicionado um nível na hierarquia anterior, colocando um agregador intermediário entre os quatro agregadores e o cliente. Essa hierarquia foi nomeada *4plus1ag1cli*.

Além de diferentes hierarquias e quantidades de coletores, foram utilizadas 4 diferentes configurações com relação à frequência de leitura dos dados e à agregação dos envios dos mesmos. Quanto à frequência, os dados foram coletados em períodos de 1 ou 0,5 segundo. No que refere-se à agregação, cada mensagem era formada pelo conteúdo de 1 ou de 5 coletas. Nos resultados utiliza-se a nomenclatura P*n*NL*m*, para identificar essas configurações. Onde *n* é o período de coleta e *m* é o número de coletas agrupadas em cada mensagem. Por exemplo, P1NL5 identifica que são coletados dados a cada 1 segundo e cada mensagem contém os dados de 5 coletas. Ou seja, uma mensagem a cada 5 segundos, aproximadamente.



(a) 1ag1cli (b) 4ag1cli

(c) 4plus1ag1cli

Figura 6.1: Hierarquias utilizadas nos experimentos

Os coletores foram executados em 80 nós de um cluster de computadores. Cada nó possui dois processadores dual core de 2.6 GHZ e 4 GB de memória. Os nós estão interconectados por uma rede Gigabit Ethernet. Esse cluster possui um serviço que permite a criação e instalação de imagens personalizadas do sistema operacional. Para os experimentos, foi modificada uma imagem padrão fornecida pela administração do sistema. Foram instaladas as bibliotecas necessárias para a compilação e execução do DIMVHCM e também o daemon de monitoramento do Ganglia.

Além disso, foram implementados scripts para automatizar a realização dos testes. A primeira tarefa realizada por esses é a instalação da imagem do sistema em todas as máquinas. Essas máquinas são alocadas utilizando um serviço de gerenciamento da alocação de recursos. Foram criados arquivos de configuração, para as diferentes distribuições dos componentes. Esses foram armazenados em uma estrutura de diretórios reconhecida pelos scripts. Para cada configuração dos coletores, é executado um script auxiliar que realiza a configuração e execução dos componentes e a coleta dos resultados. A cada execução passa-se como parâmetros, na linha de comando, o período entre coletas e o número de coletas a ser agrupado em cada mensagem.

Para cada combinação de hierarquia e número de coletores, esse script auxiliar gera os comandos a serem executados para iniciar os componentes do DIMVHCM em cada máquina. Para isso, ele lê arquivos de configuração que especificam a associação entre máquinas e componentes. Nessa etapa também são gerados arquivos de configuração

utilizados pelo coletor implementado.

A seguir, arquivos de saída gerados na etapa anterior são copiados para os nós, juntamente com um conjunto de scripts e com os arquivos do DIMVHCM. Esse deve ter sido compilado anteriormente. A seguir, em todos os nós é executado um script que executa os comandos de lançamento de cada componente e espera que sua execução termine.

A finalização da execução dos componentes é realizada, após um período de tempo pré-determinado, pelo script que está sendo executado no *frontend* do cluster. Nos testes apresentados nessa seção, esse período foi de aproximadamente 12 minutos, para cada configuração. Após isso, esse script realiza a cópia dos resultados para o *frontend*.

As três classes discutidas na seção 5.1 foram instrumentadas para registrar datas em determinados pontos. Na classe `DIMVCollector` esse registro foi feito antes de realizar a chamada ao método de recebimento de dados de um agregador remoto. Nas classes `DIMVAggregator` e `DIMVClient` foi registrada a data no início de seu método de recebimento de dados. Uma outra modificação foi que os coletores enviaram um contador de mensagens, juntamente com cada dado.

Para transmissão de dados, os componentes gravaram em arquivo o identificador do coletor que os obteve, o contador da mensagem e o timestamp citado anteriormente. Utilizando essas informações, foi possível calcular a duração de cada transmissão de dados de monitoramento. Devido à comparação de datas registradas em máquinas diferentes, foi necessário sincronizar os tempos gravados. Para isso utilizou-se um método proposto por Maillet e Tron (MAILLET; TRON, 1995).

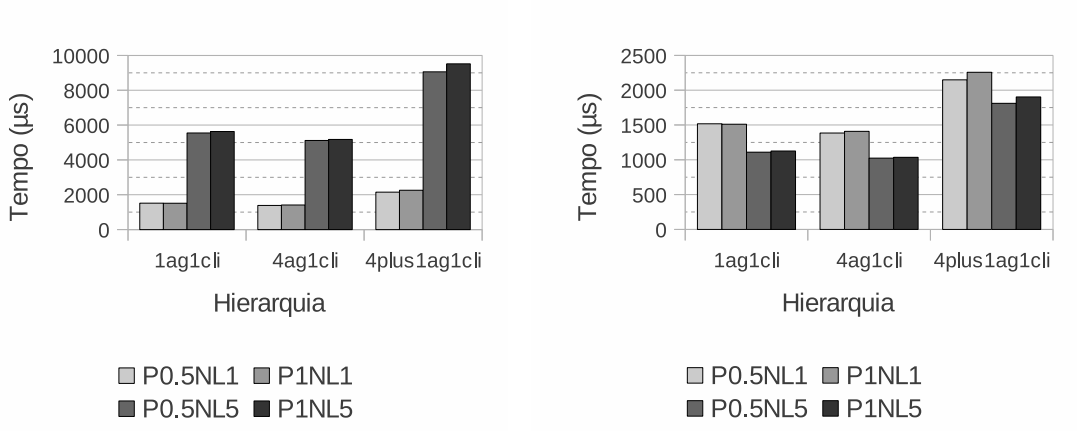
O gráfico da figura 6.2(a), mostra a média da duração do envio de uma mensagem entre coletor e cliente, utilizando 80 coletores (1 por nó). Uma versão desses resultados foi publicada anteriormente (TESSER; SCHNORR; NAVAUX, 2010). Os resultados apresentados aqui possuem melhor precisão devido a uma melhor configuração do ambiente de teste. Nota-se que os tempos para a hierarquia *1ag1cli* e *4ag1cli* são bastante similares, com uma pequena vantagem para a segunda. Como esperado, por adicionar um nível na hierarquia, a configuração *4plus1ag1cli* apresenta os piores tempos. No entanto, esse tempo não chega ao dobro dos obtidos com as outras duas hierarquias.

De maneira a visualizar melhor o efeito do agrupamento dos dados, realizou-se uma normalização dos tempos, dividindo-os pelo número de coletas contido em cada mensagem. Dessa maneira, obteve-se uma estimativa dos tempos de envio para cada coleta, a qual é mostrada na figura 6.2(b). Com 80 coletores, nota-se um ganho de aproximadamente 25% para as hierarquias *1ag1cli* e *4ag1cli* e 15% para a hierarquia *4plus1ag1cli*.

Também foram realizados testes utilizando 2 e 4 coletores por nó, ou seja 160 e 320 coletores no total. Infelizmente, durante os testes descobriu-se uma limitação na biblioteca padrão do GNUstep que não permite a criação de mais de 128 conexões para um mesmo objeto remoto. Isso impossibilitou a utilização da hierarquia *1ag1cli* com 160 e 320 coletores.

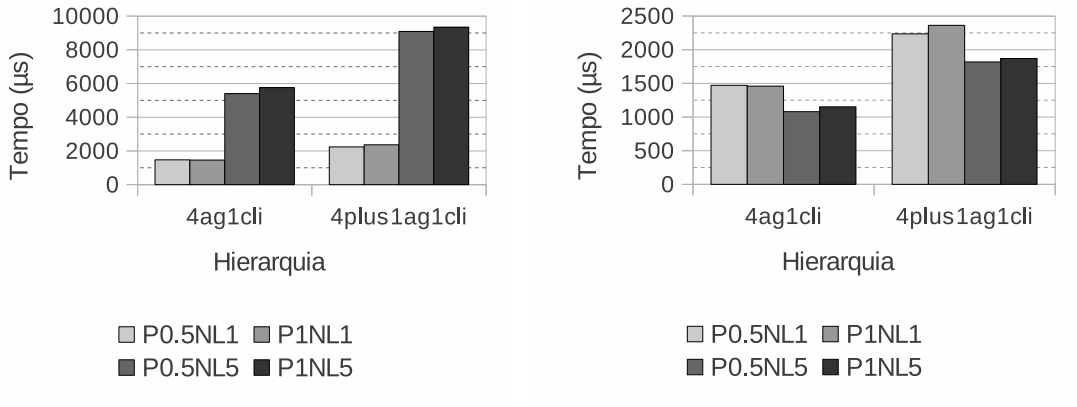
Os gráficos das figuras 6.3 e 6.4 mostram os tempos originais e normalizados, medidos utilizando 160 e 320 coletores, respectivamente. As tabela 6.1 mostra as médias e os erros encontrados no cálculo do intervalo de confiança de 95%, para todas as configurações testadas.

Com 160 coletores tem-se gráficos bastante parecidos com os para 80 coletores. Já com 320 coletores nota-se diferenças mais significativas. Destaca-se o efeito negativo da agregação de dados por mensagem com a configuração P1NL5, na hierarquia *4plus1ag1cli*. Os ganhos relativos com a agregação de dados nas mensagens (5 coletas/mensagem) são mostrados na tabela 6.2.



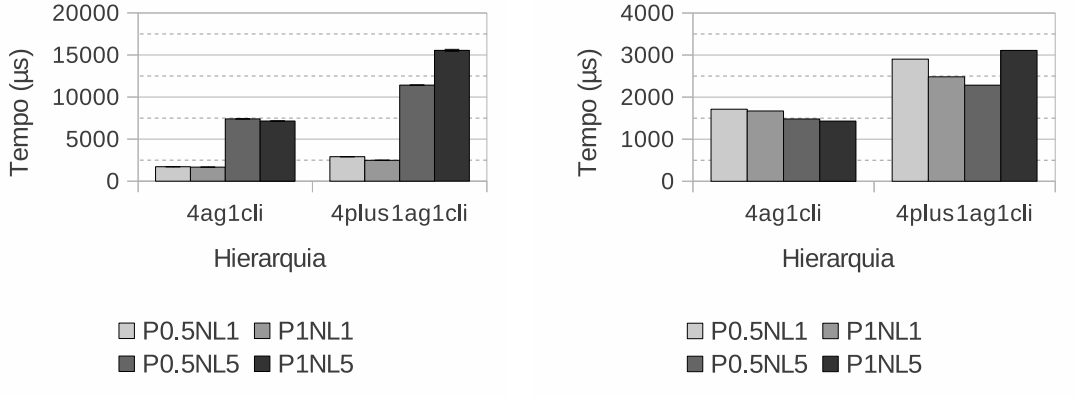
(a) Original (b) Normalizado

Figura 6.2: Duração dos envios utilizando 80 coletores



(a) Original (b) Normalizado

Figura 6.3: Duração média dos envios com 160 coletores



(a) Original (b) Normalizado

Figura 6.4: Duração média dos envios com 320 coletores

Os gráficos da figura 6.5 mostram duração média do envio de mensagens para cada

Tabela 6.1: Duração média dos envios com erros referentes ao intervalo de confiança de 95%.

Conf. dos coletores	**Hierarquia**	***1ag1cli***		***4ag1cli***		***4plus1ag1cli***	
	Coletores	***Média (µs)***	***Erro***	***Média (µs)***	***Erro***	***Média (µs)***	***Erro***
P0.5NL1	80	1516,86	2,08	1383,98	1,52	2148,25	13,01
	160			1471,62	1,32	2235,75	4,82
	320			1713,28	1,43	2903,46	4,89
P1NL1	80	1511,06	2,99	1408,60	3,48	2256,89	68,32
	160			1459,13	1,46	2362,66	29,25
	320			1671,08	1,95	2483,20	3,93
P0.5NL5	80	5546,72	16,88	5117,06	10,23	9056,80	75,07
	160			5402,04	11,24	9091,71	42,79
	320			7399,09	31,96	11416,54	42,79
P1NL5	80	5630,26	23,33	5177,01	12,79	9515,96	114,70
	160			5760,00	19,84	9350,61	64,91
	320			7146,18	34,37	15553,37	116,61

Tabela 6.2: Ganho pela agregação de dados nas mensagens (5 coletas/mensagem).

Período da coleta	**Número de coletores**	**Hierarquias**		
		1ag1cli	***4ag1cli***	***4plus1ag1cli***
0.5 s	80	26,87%	26,05%	15,68%
	160		26,58%	18,67%
	320		13,63%	21,36%
1s	80	25,48%	26,49%	15,67%
	160		21,05%	20,85%
	320		14,47%	-25,27%

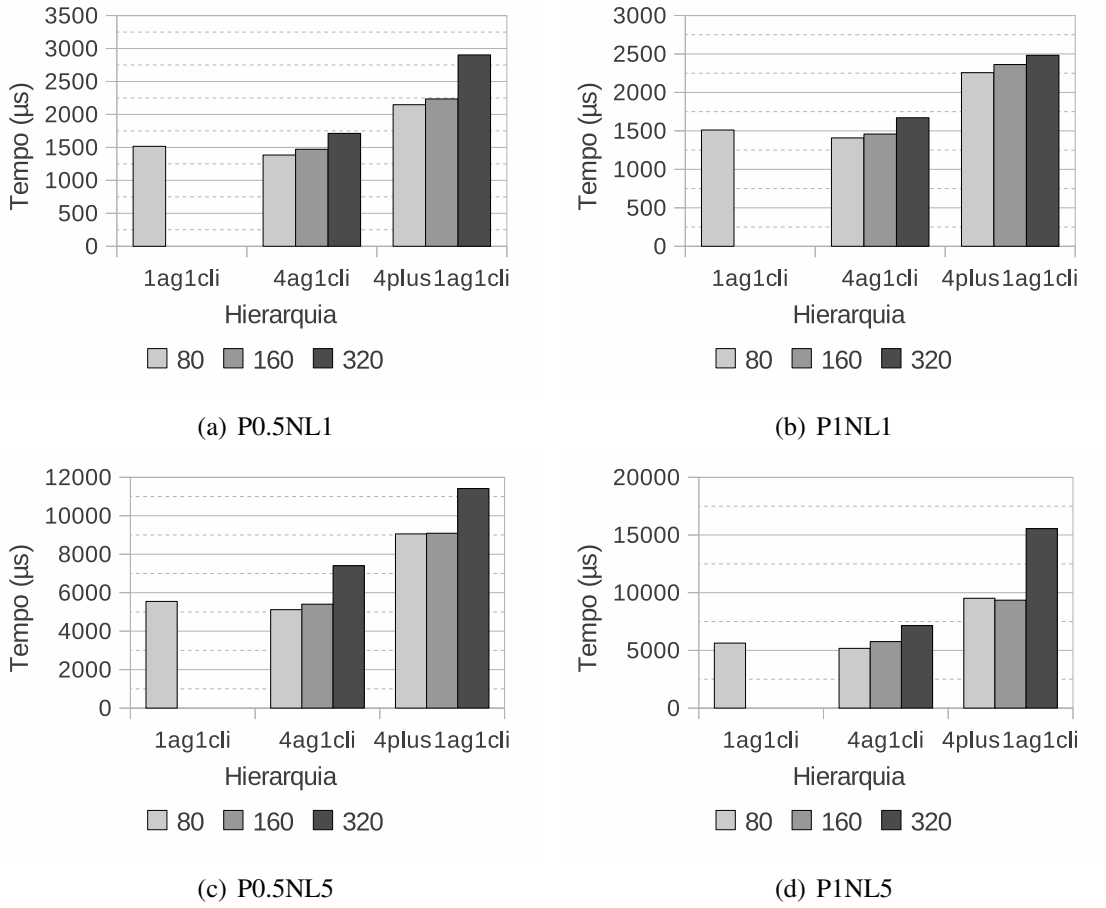
configuração de coletores, agrupados por hierarquias. Esses gráficos facilitam a comparação das diferenças de tempo entre as três quantidades de coletores utilizadas nos testes. Essas diferenças podem ser vistas também na tabela 6.3.

Para a hierarquia *4ag1cli*, a diferença máxima entre médias para 80 e 160 coletores foi de 11,26%, com a configuração P1NL5. A diferença mínima, nesse caso, foi de 3,59%, para a configuração P1NL1. Já para a hierarquia *4plus1ag1cli*, o maior aumento foi de 4,69%, com a configuração P1NL1 e o menor foi de 0,39%, com a configuração P0.5NL5.

A diferença relativa entre as médias com 160 e 320 coletores com a hierarquia *4ag1cli* varia de 14,53%, com a configuração P1NL1 até 36,97%, com a P0.5NL5. Com a hierarquia *4plus1ag1cli* essas diferenças foram de 5,10%, para a configuração P1NL1, até 66,34%, para a P1NL5. Há uma diferença negativa variando de 80 para 160 o número de coletores, com a configuração P1NL5 na hierarquia *4plus1ag1cli*. No entanto, olhando os intervalos de confiança para esses dois valores, conforme a tabela 6.1, vê-se que eles apresentam uma intersecção.

Examinando a relação entre os tempos obtidos com 80 e 160 coletores, para as hierarquias nota-se uma proximidade da variação de seus tempos com mensagens contendo dados de apenas 1 coleta. Quando envia-se dados de 5 coletas por mensagem nota-se uma perda menor de desempenho para a hierarquia *4plus1ag1cli*.

Agora, olhando a diferença entre os tempos medidos com 160 e 320 coletores, vê-se um cenário diferente. Com a configuração P0.5NL1, o sistema parece sentir o efeito do



(a) P0.5NL1

(b) P1NL1

(c) P0.5NL5

(d) P1NL5

Figura 6.5: Duração média dos envios para as quatro configurações dos coletores.

grande número de mensagens, principalmente com a hierarquia *4plus1ag1cli*. Isso se deve a essa hierarquia enviar mais mensagens, devido à adição de um nível na hierarquia. Além disso, todas as mensagens tem que ser repassadas ao cliente por um único agregador. Isso pode causar atrasos caso esse componente não consiga fazer o repasse dos dados com a mesma frequência que os recebe.

Com a configuração P1NL1, que reduz à metade a frequência de envio das mensagens, nota-se um menor aumento na duração dos envios quando se vai de 160 para 320 coletores. Nesse caso, o aumento relativo pra a hierarquia *4plus1ag1cli* chega a menos da metade do medido para a *4ag1cli*. Isso reforça a teoria de que a grande perda de desempenho com a configuração P0.5NL1 deve-se ao aumento do número de mensagens transmitidos em intervalos curtos de tempo.

Com a configuração P0.5NL5, a variação relativa entre os tempos medidos com 160 e 320 coletores foi de 36,97% para a hierarquia *4ag1cli* e 25,57% para a *4plus1ag1cli*. Ou seja, com 320 coletores, o ganho com a agregação de dados parece diminuir. No entanto, com a hierarquia *4plus1ag1cli*, a diferença relativa utilizando a configuração P0.5NL5 é menor que a obtida com a P0.5NL1, o que parece contradizer essa hipótese. Mas, quando analisamos os valores para a configuração P0.5NL1, foi observado que a hierarquia *4plus1ag1cli* é mais afetada pela maior frequência das mensagens, que a *4ag1cli*. A frequência dos envios na configuração P0.5NL5 é um quinto daquela gerada pela P0.5NL1. Portanto, para a hierarquia *4plus1ag1cli*, a perda de desempenho com o aumento do tamanho das mensagens é mascarada pelo ganho devido à diminuição da

Tabela 6.3: Diferenças de tempo com as diferentes quantias de coletores.

Conf. dos coletores	**Número de coletores**	**Hierarquias**	
		4ag1cli	***4plus1ag1cli***
P0.5NL1	80 para 160	6,33%	4,07%
	160 para 320	16,42%	29,87%
	80 para 320	23,79%	35,15%
P1NL1	80 para 160	3,59%	4,69%
	160 para 320	14,53%	5,10%
	80 para 320	18,63%	10,03%
P0.5NL5	80 para 160	5,57%	0,39%
	160 para 320	36,97%	25,57%
	80 para 320	44,60%	26,05%
P1NL5	80 para 160	11,26%	-1,74%
	160 para 320	24,07%	66,34%
	80 para 320	38,04%	63,45%

frequência com que elas são enviadas.

As diferenças relativas entre as medidas obtidas com 160 e 320 coletores utilizando a configuração P1NL5 é de 24,07% para a hierarquia *4ag1cli* e 66,34% para a *4plus1ag1cli*. Na hierarquia *4ag1cli*, o impacto dessa duplicação do número de coletores foi menor com essa configuração que com a P0.5NL5. Isso deve-se à diminuição da frequência das mensagens. No entanto, o contrário acontece com a hierarquia *4plus1ag1cli*.

A tabela 6.4 mostra as diferenças relativas entre tempos medidos para as três hierarquias avaliadas. É interessante notar as variações das médias entre as hierarquias *4ag1cli* e *4plus1ag1cli*. Quando se varia de 80 para 160 coletores, há uma diminuição nas diferenças entre essa duas hierarquias, com as configurações P0.5NL1, P0.5NL5 e P1NL5. Com a P1NL1, há um pequeno aumento da diferença (1,70%). Um cenário mais interessante ocorre quando se varia a quantidade de coletores de 160 para 320. Nesse caso, com a configuração P0.5NL1, há um aumento na diferença, que vai de 51,92% para 69,47%. Isso deve-se à sensibilidade da hierarquia *4plus1ag1cli* ao aumento do número de mensagens, conforme discutido anteriormente. Agora, observando as diferenças para a configuração P1NL1, vê-se que a diminuição das mensagens inverteu o quadro. Ou seja, houve uma diminuição da diferença entre as duas hierarquias. Com a configuração P0.5NL5, a diferença relativa entre as hierarquias *4ag1cli* e *4plus1ag1cli* diminui quando se vai de 160 para 320 coletores. Isso assemelha-se ao que acontece com a configuração P1NL1. Já com a configuração P1NL5, quando se vai de 160 a 320 coletores, há um grande aumento na diferença entre as duas hierarquias. Isso deve-se principalmente ao baixo desempenho apresentado pela hierarquia 4plus1ag1cli, nessa configuração (P1NL5 com 320 coletores).

Além da duração total, foi calculada a duração dos envios de dados de monitoramento entre os níveis das hierarquias do DIMVHCM. Utilizando esses resultados, pode-se observar como essas etapas contribuem para o tempo total. É importante ressaltar que as durações medidas incluem o tempo em que as chamadas ficam esperando para serem executadas por um `RunLoop`, no objeto receptor. Ou seja, esse tempo é maior que o tempo necessário para transmitir os dados pela rede.

Obtidas essas durações parciais, foi calculado seu custo relativo à duração total dos envios. A partir desses, gerou-se gráficos em que se pode observar a contribuição de cada

Tabela 6.4: Diferenças de tempo entre as diferentes hierarquias.

Conf. Dos coletores	Núm. de coletores	*1ag1cli para 4ag1cli*	*4ag1cli para 4plus1ag1cli*	*1ag1cli para 4plus1ag1cli*
P0.5NL1	80	-8,76%	55,22%	41,63%
	160		51,92%	
	320		69,47%	
P1NL1	80	-6,78%	60,22%	49,36%
	160		61,92%	
	320		48,60%	
P0.5NL5	80	-7,75%	76,99%	63,28%
	160		68,30%	
	320		54,30%	
P1NL5	80	-8,05%	83,81%	69,01%
	160		62,34%	
	320		117,65%	

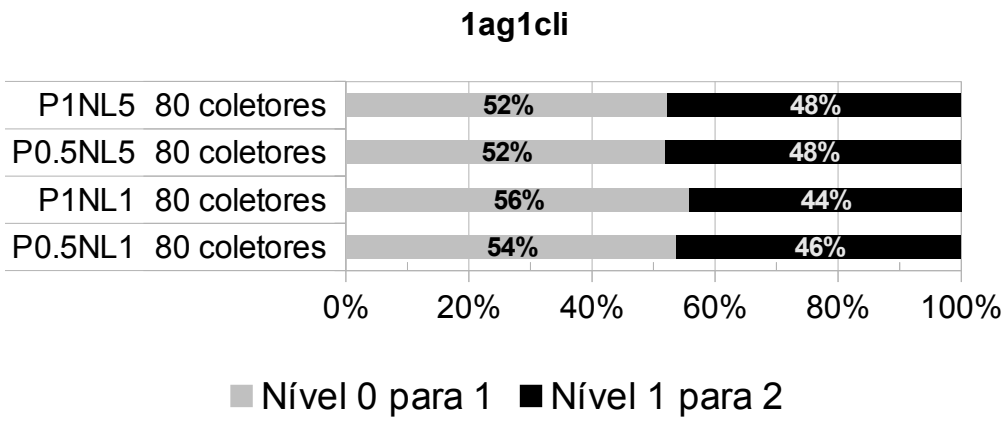


Figura 6.6: Tempo para envio de dados entre os níveis da hierarquia 1ag1cli

etapa no tempo total. O primeiro desses gráficos é mostrado na figura 6.6 e refere-se à hierarquia *1ag1cli*. Nessa, nota-se que os tempos em cada etapa são bastante similares, sendo um pouco maiores no envio entre coletores e agregador.

Na figura 6.7, temos a porcentagem de tempo em cada etapa da transmissão dos dados na hierarquia *4ag1cli*. Nesse caso, observa-se resultados similares aos do gráfico anterior. Ou seja, o tempo necessário para transmitir dados do coletor para os agregadores foi ligeiramente maior que o transcorrido na etapa seguinte.

Os percentuais para a hierarquia *4plus1ag1cli* são mostrados no gráfico da figura 6.8. Primeiramente nota-se que o envio dos dados entre os níveis 1 e 2 ocupou parcelas similares do tempo total. Essa característica repete-se nos outros níveis, sendo exceção o teste com a configuração P1NL5, com 320 coletores. Nesse caso a demora foi visivelmente maior nas duas últimas etapas do envio.

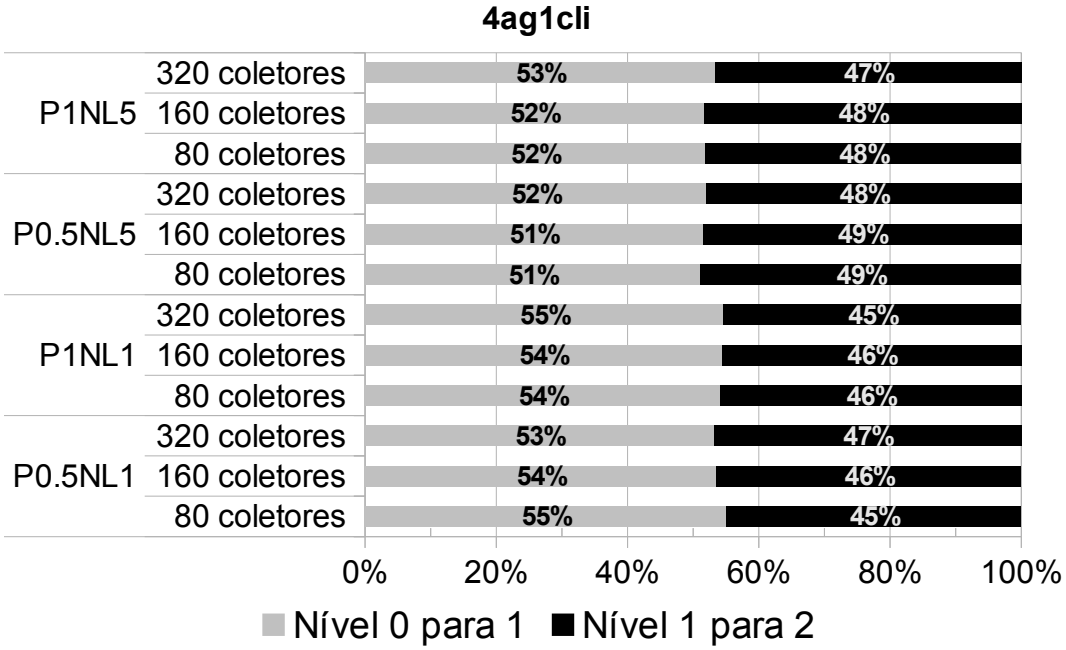


Figura 6.7: Tempo para envio de dados entre os níveis da hierarquia 4ag1cli

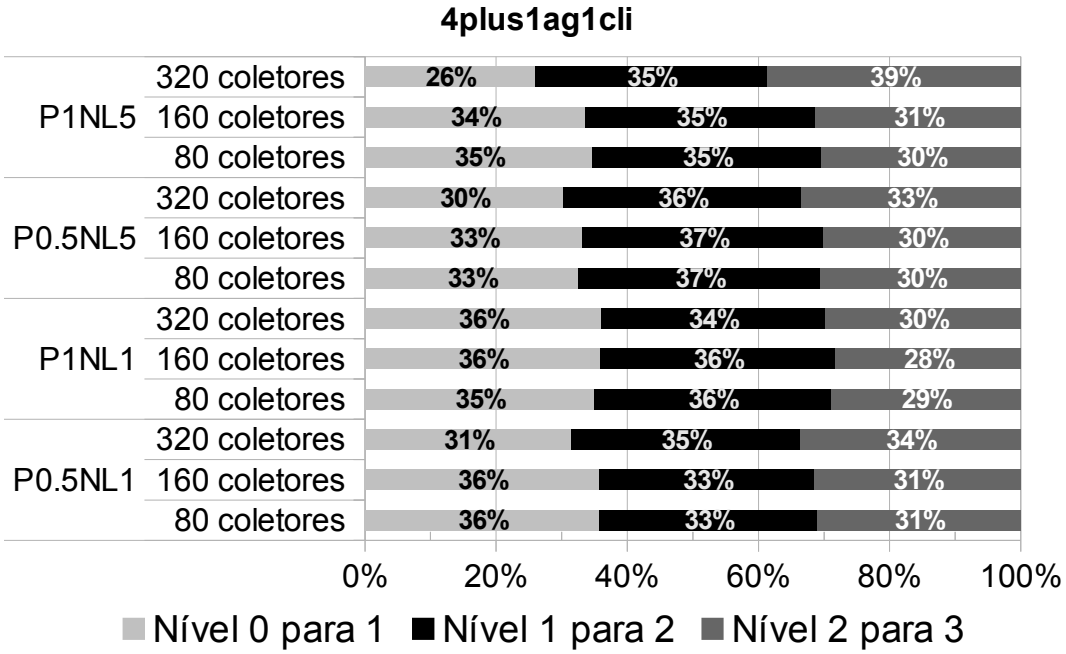


Figura 6.8: Tempo para envio de dados entre os níveis da hierarquia 4plus1ag1cli

6.2 Avaliação do processamento dos dados no cliente

Para avaliar o processamento dos dados no cliente, o DIMVisual e o TRIVA foram instrumentados para registrar alguns eventos. No DIMVisual foram registrados os seguintes eventos:

- recebimento de um dado do `DIMVClient` pelo *data source*;

- consumo do dado pelo *data source*, ou seja, o início do processo de geração de eventos a partir do dado;
- leitura de eventos, ou seja, quando um evento é gerado a partir de dados recebidos do DIMVHCM;
- fim da conversão de um evento, que é feita quando requisitada pelo *DIMVHCM Reader*.

No TRIVA os eventos registrados foram:

- recebimento, pelo *DIMVHCM Reader*, de um evento convertido;
- consumo de um evento pelo *DIMVHCM Reader*;
- leitura de um *chunk* completo pelo *DIMVHCM Reader*, incluindo o repasse dos dados para o componente seguinte do TRIVA (`PajeEventDecoder`).
- atualização da visualização.

Os experimentos com o DIMVHCM foram executados em um cluster geograficamente distante de onde o pesquisador situava-se. Por isso, não foi possível conectar diretamente o cliente que utiliza a ferramenta de visualização ao DIMVHCM. Para contornar essa dificuldade, primeiramente foi implementado um cliente escritor, que grava os dados recebidos em um arquivo. Esse cliente também registra a data em que cada dado é recebido, juntamente com seu tamanho. Depois, foi implementado um programa que imita o comportamento do DIMVHCM. Esse programa foi chamado `HCMSimulator`.

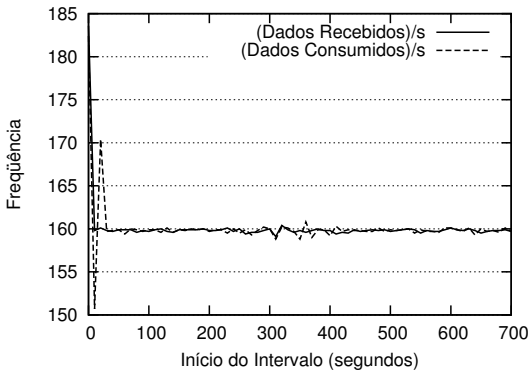
O `HCMSimulator` lê os dados gravados pelo cliente escritor e os fornece ao cliente visualizador. Esse fornecimento é cronometrado utilizando as datas registradas pelo cliente escritor. Dessa forma é possível fornecer localmente os dados ao cliente visualizador, imitando o fornecimento *on-line* realizado pelo DIMVHCM.

Primeiramente foram coletados os dados de monitoramento no cluster, utilizando o cliente escritor para receber dados de 80 coletores. Para esse experimento foi utilizada a hierarquia *4ag1cli*, que é a que apresentou melhor desempenho. A configuração utilizada para os coletores foi a P0.5NL1, devido a ser a que apresenta maior frequência de mensagens, dentre as avaliadas.

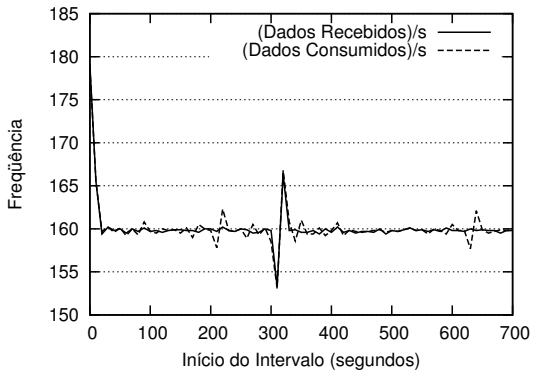
Após a coleta dos dados, utilizou-se o `HCMSimulator`, para realizar a avaliação do cliente. O cliente foi executado com três diferentes tamanhos para o *time-slice* de integração do TRIVA: 5, 10 e 15 segundos. A máquina utilizada para isso possui um processador Pentium D de 3 GHz e 1 GB de memória DDR2. Os eventos registrados foram divididos em fatias de 10 segundos. Para cada fatia foi calculada a frequência com que cada um dos eventos, listados no início dessa sessão, ocorreram. Finalmente, foram gerados gráficos comparando frequências dos pares de eventos que ocorrem em sequência. Dessa forma pode-se analisar se as informações estão sendo processadas na mesma frequência em que estão sendo geradas.

Os gráficos da figura 6.9 mostram as frequências de recebimento de dados pelo DIMVisual e de seu consumo. Observando-os não é possível notar disparidades significativas entre o desempenho dessas duas etapas, com quaisquer dos intervalos de integração.

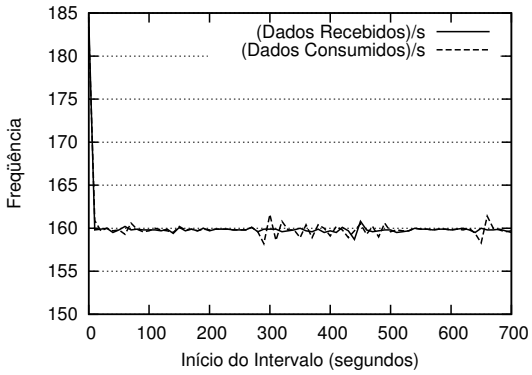
A seguir, tem-se a figura 6.10, que compara as frequências de geração de eventos e de sua conversão para o formato Pajé. Nesses gráficos pode-se notar que com uma fatia de



(a) *Time-slice*: 5s



(b) *Time-slice*: 10s



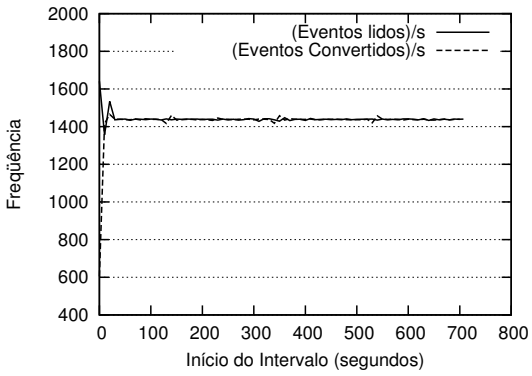
(c) *Time-slice*: 15s

Figura 6.9: DIMVisual: Frequências de Recebimento x Consumo de dados

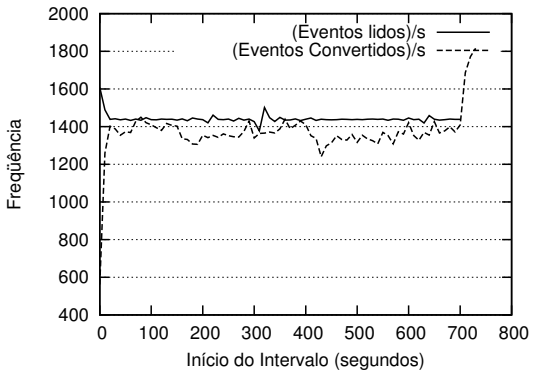
tempo de 5 segundos, as frequências são similares. No entanto, quando se aumenta o *time-slice*, a conversão dos eventos passa a apresentar um atraso. Isso significa que o aumento do uso do processador pelo TRIVA está afetando o desempenho de outros componentes. Os atrasos podem estar tanto no DIMVisual, quanto no *DIMVHCM Reader*, devido a os dados só serem convertidos quando requisitados por esse último. Esse atraso poderia ser diminuido ou até eliminado através do uso de um processador de maior desempenho. A utilização de um formato de dados mais eficiente que o XML também poderia melhorar o desempenho da conversão. Além disso, o TRIVA sempre integra os valores de todo intervalo. Isso não é necessário quando o novo intervalo intercepta o anterior. Pretende-se modificar o TRIVA para que ele calcule apenas as integrais das partes diferentes dos dois intervalos. Somando a integral anterior à do intervalo adicionado e subtraindo a do removido obtém-se a integral do novo intervalo.

A próxima comparação realizada foi entre a frequência de recebimento e consumo de eventos dentro do *DIMVHCM Reader*. Como se vê nos gráficos da figura 6.11, não há diferença significativa entre as duas. Isso é um resultado positivo, pois indica componente conseguiu repassar os dados numa frequência similar à de seu recebimento.

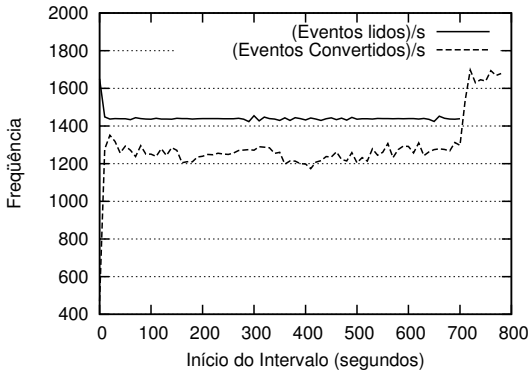
Por último, na figura 6.12, temos a comparação da frequência com que o *DIMVHCM Reader* repassa *chunks* com a que a visualização é atualizada. Da maneira como foi implementada a atualização *on-line* da visualização, inicia-se uma atualização a cada vez que um novo *chunk* é lido. Pelos gráficos nota-se uma similaridade entre as duas frequências. É interessante notar que nesse caso não é tão importante que a tela seja atualizada com a mesma frequência que os dados. Os atrasos não são cumulativos como nos casos ante-



(a) *Time-slice*: 5s



(b) *Time-slice*: 10s



(c) *Time-slice*: 15s

Figura 6.10: DIMVisual: Frequências de Leitura x Conversão de Eventos

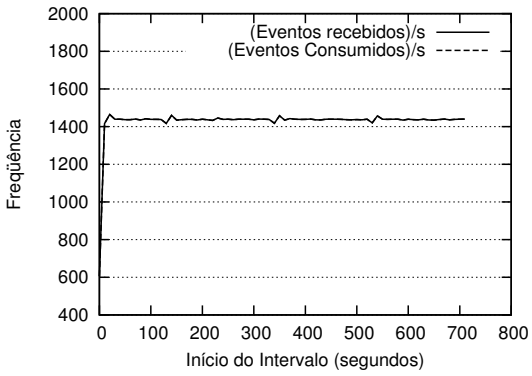
riores. O quanto de atraso pode ser tolerado depende do quão atualizada a visualização precisa estar para possibilitar a detecção do fenômeno que o usuário quer observar.

6.3 Considerações finais

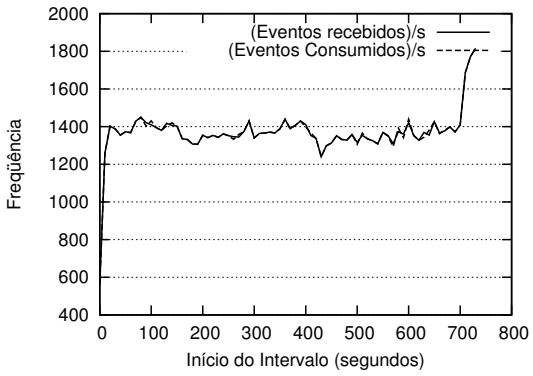
Nesse capítulo foram apresentados experimentos realizados para avaliar o DIMVHCM. Esses incluíram testes envolvendo o envio de mensagens através da hierarquia e também a avaliação da capacidade de processamento dos dados pelo cliente implementado.

Foram obtidas medidas do tempo necessário para mensagens irem de um coletor até um cliente, utilizando diferentes hierarquias e configurações do coletor. Em todos os resultados a média dos tempos a medidos foi consideravelmente menor que os intervalos entre o envio das mensagens (aproximadamente 0,5% no pior caso) , o que é bastante promissor. Dentre as hierarquias testadas, aquela que possui um nível a mais, composto por um único agregador, mostrou-se a menos eficiente e também a mais vulnerável ao aumento do número de mensagens. Além disso, foi mostrado ser mais eficiente ter-se mais de um agregador por nível da hierarquia. Também foi observado o ganho obtido ao agregar dados de múltiplas coletas em cada mensagem, em contraste com enviar uma mensagem para cada coleta.

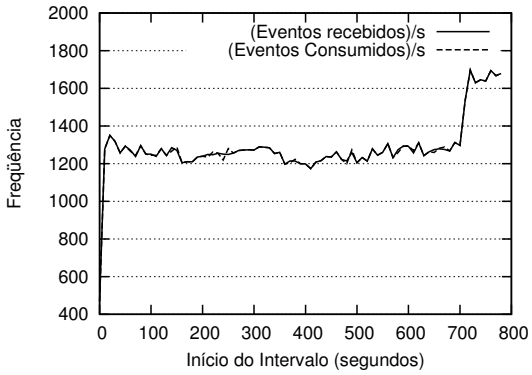
Os testes de desempenho do cliente consistiram em comparar a frequência com que certos eventos ocorrem dentro do cliente. Dentre os eventos observados, foram comparados aqueles pares que ocorrem em sequência. Essas medidas foram obtidas enquanto o TRIVA gerava a visualização utilizando três diferentes intervalos (*time-slices*) de integra-



(a) *Time-slice*: 5s



(b) *Time-slice*: 10s

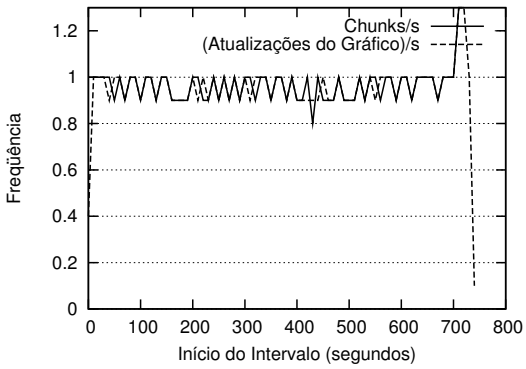


(c) *Time-slice*: 15s

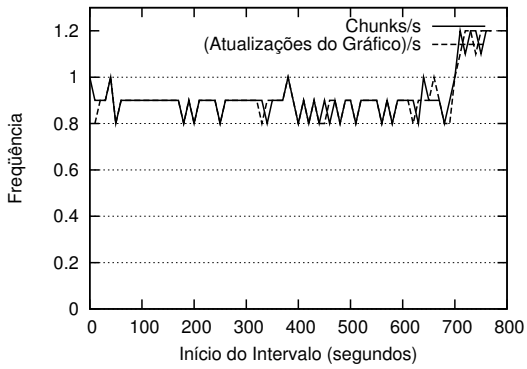
Figura 6.11: DIMVHCM Reader: Frequências de Recebimento x Consumo de Eventos

ção: 5, 10 e 15 segundos. O cliente mostrou um bom desempenho para o *time-slice* de 5 segundos. Já para o de 10 e o de 15 segundos, houve um atraso na conversão de eventos. Isso indica que o maior uso de CPU, para cálculo da integração dos valores no intervalo de tempo, interferiu na conversão. Nesse caso, pode ter ocorrido perda de desempenho tanto na thread produtora do *DIMVHCM Reader* quanto na própria conversão de eventos pelo DIMVisual. Esse atraso pode ser diminuído com o uso de um processador de maior desempenho. Além disso foi proposta uma modificação na integração de valores do *time-slice*, pelo TRIVA. Com essa, o desempenho dessa tarefa deve melhorar, diminuindo a interferência com outra partes do programa.

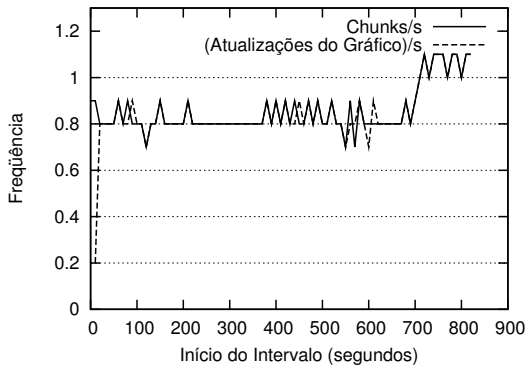
No capítulo a seguir temos a conclusão desta dissertação. Além disso, serão apresentados tópicos que podem ser explorados no futuro, dando continuação a esse trabalho.



(a) *Time-slice*: 5s



(b) *Time-slice*: 10s



(c) *Time-slice*: 15s

Figura 6.12: TRIVA: Frequências de Leitura de *Chunks* x Atualização da Visualização

7 CONCLUSÕES E TRABALHOS FUTUROS

Nesta dissertação foi proposto um modelo de coleta de dados de monitoramento. Esse tem como objetivo fornecer dados para a análise *on-line* do comportamento de aplicações e sistemas distribuídos. No âmbito do trabalho decidiu-se focar na análise através de visualização. Esse objetivo e o uso de visualização diferenciam esse trabalho de outras ferramentas preexistentes, que possuem características similares.

Para atingir o objetivo proposto, foi elaborado um modelo hierárquico para coleta *on-line* de dados de monitoramento distribuídos. Esse modelo foi pensado para ser integrado ao DIMVisual, que é um modelo de integração de dados de monitoramento para visualização. Devido a essa integração, o modelo foi batizado "*DIMVisual Hierarchical Collection Model*" (DIMVHCM).

Por ser hierárquico, ele possibilita acessar informações sobre um conjunto de objetos a partir de um único ponto. Uma vantagem adicional é a adequação à estrutura física hierárquica apresentada por vários sistemas distribuídos. Outra decisão foi o uso do modelo *push* de recebimento de dados, que é mais adequado para a análise comportamental. Além disso, foi incluído um mecanismo de subscrição. Dessa forma possibilita-se a escolha de um conjunto de informações a ser transmitido. Por consequência, diminui-se o uso de recursos para coleta de informações desnecessárias.

Através desse conjunto de características, obteve-se um modelo capaz de coletar dados de monitoramento distribuídos de maneira *on-line*. Essa capacidade foi comprovada através da utilização de um protótipo do DIMVHCM para coletar dados de monitoramento dos nós de um cluster de computadores.

Esse protótipo foi desenvolvido na linguagem Objective-C. Após, ele foi integrado com os protótipos do DIMVisual e do TRIVA. Assim obteve-se uma ferramenta de monitoramento *on-line* completa, que realiza a geração, distribuição, processamento e apresentação dos dados de monitoramento. Essa ferramenta satisfaz o objetivo citado no início deste capítulo. Ou seja, ela utiliza dados fornecidos pelo DIMVHCM para gerar uma visualização on-line, através da qual pode-se analisar o comportamento de aplicativos e sistemas distribuídos. Essa capacidade foi testada através de sua utilização para visualização *on-line* de dados coletados de um cluster de computadores.

Em comparação realizada com um conjunto de ferramentas de monitoramento, esse protótipo é o que possui o melhor conjunto de características com relação à apresentação (visualização) das informações. Isso é importante, pois faz parte de seu objetivo utilizar visualização para acompanhar o comportamento de aplicações e programas. Além disso, segundo a classificação de Zanikolas e Sakellariou, o protótipo é um sistema de nível 3, genérico e empilhável. Essa é a melhor classificação possível na taxonomia.

O protótipo desenvolvido foi utilizado na realização de experimentos para avaliar o modelo. O primeiro desses avaliou o tempo necessário para a transmissão de dados de

monitoramento. O segundo avaliou a capacidade do cliente processar os dados recebidos.

No primeiro experimento, foram utilizadas três diferentes hierarquias. Dados foram coletados a cada 0.5 ou 1 segundo e enviados a cada 1 ou 5 coletas. Os coletores foram executados em 80 nós de um cluster. Foram medidos os tempos com 1, 2 e 4 coletores por nó, resultando em um total de 80, 160 ou 320 coletores, respectivamente. Os dados utilizados foram obtidos de daemons do Ganglia (*gmond*), no formato XML. Cada coleta obteve aproximadamente 7 KB de dados.

Os resultados desse teste foram promissores, pois as médias das durações dos envios foram bastante inferiores ao intervalo entre eles. A maior média dos tempos medidos foi de aproximadamente 1,6 centésimos de segundo. Além disso, foram medidos os tempos para o envio entre cada dois níveis da hierarquia. Com esses dados comparou-se a contribuição de cada etapa para a duração total dos envios. As médias dos tempos para cada etapa ficaram próximas de 50% das médias de tempo para o envio completo, nas hierarquias com 3 níveis. Na hierarquia *4plus1ag1cli*, que tem 4 níveis, as médias para cada etapa ficaram próximas de 30% das médias de tempo para o envio completo. Esses resultados mostram que há uma boa distribuição dos tempos entre as etapas.

Para avaliar o desempenho do cliente, registrou-se a ocorrência de 8 tipos de evento durante o processamento dos dados. Então, foram comparadas as frequências com que pares de eventos consecutivos ocorreram. Os dados foram obtidos a cada 0.5 segundo e enviados a cada coleta. O TRIVA foi executado utilizando três intervalos de integração: 5, 10 e 15 segundos.

O cliente conseguiu processar os dados recebidos sem atrasos significativos na maioria das etapas observadas. A exceção foi a conversão de eventos pelo DIMVisual, quando aumentado o intervalo de integração (*time-slice*) do TRIVA, de 5 para 10 ou 15 segundos. O atraso detectado nessas configurações é causado pelo aumento do processamento necessário nessa integração. Esse atraso pode ser diminuído, ou até eliminado, através da utilização de uma CPU de maior desempenho. Além disso foi proposta uma modificação na integração de dados realizada pelo TRIVA, a qual deve melhorar o desempenho dessa tarefa. Outro fator que pode melhorar o desempenho da fonte de dados do DIMVisual, seria o uso de um formato de dados mais eficiente que XML.

Como trabalho futuro, um aspecto a ser estudado é a intrusividade causada pelo sistema de monitoramento. Também seria interessante realizar avaliações do DIMVHCM com diferentes tamanhos de mensagem. Além disso, podem ser apresentados exemplos práticos de uso do modelo. Outro trabalho interessante seria adicionar mecanismos de preprocessamento dos dados aos agregadores. Isso seria uma tentativa de aumentar a escalabilidade do sistema.

A implementação do protótipo também pode ser aprimorada. Para aumentar o desempenho, pode-se explorar a criação de mais de uma conexão dos agregadores com os componentes superiores. Dessa maneira pode haver uma diminuição no tempo que as mensagens ficam esperando para serem repassadas. Para suportar essa modificação seria utilizado um modelo produtor-consumidor dentro dos agregadores. Outra possibilidade seria a utilização de outros mecanismos de comunicação. Um exemplo seria um middleware orientado a mensagens. Além disso, seria importante a adição da capacidade de comunicação interdomínios, que não é suportada pelos objetos distribuídos do GNUstep. Nesse caso, seria necessário lidar com dificuldades como firewalls e o uso de *Network Address Translation* (NAT) no sistema.

REFERÊNCIAS

AGARWALA, S. et al. System-Level Resource Monitoring in High-Performance Computing Environments. **J. Grid Comput.**, [S.l.], v.1, n.3, p.273–289, 2003.

AIFTIMIEI, C. et al. Recent evolutions of GridICE: a monitoring tool for grid systems. In: GMW '07: PROCEEDINGS OF THE 2007 WORKSHOP ON GRID MONITORING, New York, NY, USA. **Anais...** ACM, 2007. p.1–8.

ANDREOZZI, S. et al. GridICE: a monitoring service for grid systems. **Future Gener. Comput. Syst.**, Amsterdam, The Netherlands, The Netherlands, v.21, n.4, p.559–571, 2005.

CHANDY, K. M.; LAMPORT, L. Distributed Snapshots: determining global states of distributed systems. **ACM Transactions on Computer Systems**, [S.l.], v.3, p.63–75, 1985.

CZAJKOWSKI, K. et al. Grid Information Services for Distributed Resource Sharing. In: HPDC '01: PROCEEDINGS OF THE 10TH IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, Washington, DC, USA. **Anais...** IEEE Computer Society, 2001. p.181.

FOSTER, I. **A Globus Toolkit Primer**. Disponível em: <http://globus.org/toolkit/docs/4.0/key/GT4_Primer_0.6.pdf>. Acesso em Julho 2011.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: enabling scalable virtual organizations. **International Journal of Supercomputer Applications**, [S.l.], v.15, n.3, 2001.

FOSTER, I. T. Globus Toolkit Version 4: software for service-oriented systems. In: NPC. **Anais...** Springer, 2005. p.2–13. (Lecture Notes in Computer Science, v.3779).

GUNTER, D. et al. NetLogger: a toolkit for distributed system performance analysis. In: MASCOTS '00: PROCEEDINGS OF THE 8TH INTERNATIONAL SYMPOSIUM ON MODELING, ANALYSIS AND SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS, Washington, DC, USA. **Anais...** IEEE Computer Society, 2000. p.267.

GUNTER, D. et al. On-Demand Grid Application Tuning and Debugging with the NetLogger Activation Service. In: GRID '03: PROCEEDINGS OF THE FOURTH INTERNATIONAL WORKSHOP ON GRID COMPUTING, Washington, DC, USA. **Anais...** IEEE Computer Society, 2003. p.76.

HOLLINGSWORTH, J.; TIERNEY, B. Instrumentation and Monitoring. In: FOSTER, I.; KESSELMAN, C. (Ed.). **The Grid 2**: blueprint for a new computing infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

JANCIC, J. et al. dproc - Extensible Run-Time Resource Monitoring for Cluster Applications. In: ICCS '02: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE-PART II, London, UK. **Anais...** Springer-Verlag, 2002. p.894–903.

JOYCE, J. et al. Monitoring distributed systems. **ACM Trans. Comput. Syst.**, New York, NY, USA, v.5, n.2, p.121–150, 1987.

KERGOMMEAUX, J. de; OLIVEIRA STEIN, B. de. Pajé: an extensible environment for visualizing multi-threaded programs executions. In: BODE, A. et al. (Ed.). **Euro-Par 2000 Parallel Processing**. [S.l.]: Springer Berlin / Heidelberg, 2000. p.133–140. (Lecture Notes in Computer Science, v.1900). 10.1007/3-540-44520-X_17.

KUTARE, M. et al. Monalytics: online monitoring and analytics for managing large scale data centers. In: PROCEEDING OF THE 7TH INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING, New York, NY, USA. **Anais...** ACM, 2010. p.141–150. (ICAC '10).

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

LEGRAND, I. C. et al. MonALISA: an agent based, dynamic service system to monitor, control and optimize grid based applications. In: COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS CONFERENCE (CHEP2004), Interlaken, Switzerland. **Anais...** IEEE Computer Society, 2004. p.181.

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

Mansouri-Samani and Morris Sloman, M. **Monitoring Distributed Systems (A Survey)**. London, UK: Imperial College of Science Technology and Medicine, 1995. (DOC92/23).

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The Ganglia Distributed Monitoring System: design, implementation, and experience. **Parallel Computing**, [S.l.], v.30, 2004.

MILLER, B. P. et al. The Paradyn Parallel Performance Measurement Tool. **Computer**, Los Alamitos, CA, USA, v.28, n.11, p.37–46, 1995.

NEWMAN, H. B. et al. MonALISA: a distributed monitoring service architecture. In: COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS CONFERENCE (CHEP2003), La Jola, California. **Anais...** [S.l.: s.n.], 2003.

NEWMAN, H. B.; LEGR, I. C.; BUNN, J. J. A Distributed Agent-based Architecture for Dynamic Services. In: CHEP 2001, BEIJING, SEPT 2001, HTTP://CLEGRAND.HOME.CERN.CH/CLEGRAND/CHEP01/CHE. **Anais...** [S.l.: s.n.], 2001. p.01–10.

RIBLER, R. L. et al. Autopilot: adaptive control of distributed applications. In: IEEE SYMPOSIUM ON HIGH-PERFORMANCE DISTRIBUTED COMPUTING, 7. **Proceedings...** [S.l.: s.n.], 1998. p.172–179.

RUSSELL, R. D.; CHAVAN, M. Fast Kernel Tracing: a performance evaluation tool for linux. In: IASTED INTERNATIONAL CONFERENCE ON APPLIED INFORMATICS (AI 2001), 19. **Proceedings...** [S.l.: s.n.], 2001.

SACERDOTI, F. D. et al. Wide area cluster monitoring with Ganglia. In: IEEE INTERNATIONAL CONFERENCE ON CLUSTER COMPUTING. PROCEEDINGS., 2003. **Anais...** [S.l.: s.n.], 2003. p.289–298.

SCHNORR, L. M.; HUARD, G.; NAVAUX, P. 3D approach to the visualization of parallel applications and Grid monitoring information. In: GRID '08: PROCEEDINGS OF THE 2008 9TH IEEE/ACM INTERNATIONAL CONFERENCE ON GRID COMPUTING, Washington, DC, USA. **Anais...** IEEE Computer Society, 2008. p.233–241.

SCHNORR, L. M.; HUARD, G.; NAVAUX, P. O. A. Towards Visualization Scalability through Time Intervals and Hierarchical Organization of Monitoring Data. In: CCGRID '09: PROCEEDINGS OF THE 2009 9TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, Washington, DC, USA. **Anais...** IEEE Computer Society, 2009. p.428–435.

SCHNORR, L. M.; HUARD, G.; NAVAUX, P. O. A. Triva: interactive 3d visualization for performance analysis of parallel applications. **Future Gener. Comput. Syst.**, Amsterdam, The Netherlands, The Netherlands, v.26, n.3, p.348–358, 2010.

SCHNORR, L. M.; NAVAUX, P. O. A.; HUARD, G. Visual Mapping of Program Components to Resources Representation: a 3d analysis of grid parallel applications. In: SBAC-PAD '09: PROCEEDINGS OF THE 2009 21ST INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, Washington, DC, USA. **Anais...** IEEE Computer Society, 2009. p.127–134.

SCHNORR, L. M.; NAVAUX, P. O. A.; OLIVEIRA STEIN, B. de. DIMVisual: data integration model for visualization of parallel programs behavior. In: CLUSTER COMPUTING AND THE GRID, 2006. CCGRID 06. SIXTH IEEE INTERNATIONAL SYMPOSIUM ON, Los Alamitos, CA, USA. **Anais...** IEEE Computer Society, 2006. v.00, p.473–480.

TESSER, R. K.; NAVAUX, P. O. A. A Hierarchical Model for Distributed Monitoring Data Collection. In: WSPPD 2008: WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUíDO. **Anais...** [S.l.: s.n.], 2008.

TESSER, R. K.; NAVAUX, P. O. A. Um modelo hierárquico para coleta de dados de monitoramento em sistemas distribuídos. In: IX ESCOLA REGIONAL DE ALTO DESEMPENHO. **Anais...** [S.l.: s.n.], 2009. p.131–132.

TESSER, R. K.; SCHNORR, L. M.; NAVAUX, P. O. A. Enabling Online Visualization Through Distributed Trace Collection. In: WSPPD 2009: WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUíDO. **Anais...** [S.l.: s.n.], 2009.

TESSER, R. K.; SCHNORR, L. M.; NAVAUX, P. O. A. On-line Collection of Monitoring Data in Distributed Systems Using a Hierarchical Structure. In: LATIN AMERICAN CONFERENCE ON HIGH PERFORMANCE COMPUTING, 3. **Proceedings...** [S.l.: s.n.], 2010. p.133–140.

The Globus Alliance. **Página do Globus Toolkit**. Disponível em: <http://www.globus.org/toolkit/>. Acesso em: Julho 2011.

TIERNEY, B. et al. **A Grid Monitoring Architecture**. Disponível em: <http://www.ogf.org/documents/GFD.7.pdf>. Acesso em: Julho 2011.

Tobias Oetiker. **Web site do RRDtool**. Disponível em: <http://oss.oetiker.ch/rrdtool/>. Acesso em: Julho 2011.

WOLSKI, R.; SPRING, N. T.; HAYES, J. The network weather service: a distributed resource performance forecasting service for metacomputing. **Future Generation Computer Systems**, [S.l.], v.15, n.5–6, p.757–768, 1999.

ZANIKOLAS, S.; SAKELLARIOU, R. A Taxonomy of Grid Monitoring Systems. **Future Generation Computer Systems**, [S.l.], v.21, n.1, p.163–188, January 2005.

ZHANG, X.; FRESCHL, J. L.; SCHOPF, J. M. Scalability analysis of three monitoring and information systems: mds2, r-gma, and hawkeye. **J. Parallel Distrib. Comput.**, Orlando, FL, USA, v.67, n.8, p.883–902, 2007.