Elizeu Elieber Fachini

Um Sistema de Monitoramento para Caracterização de Algoritmos Distribuídos

Sorocaba, SP

Fevereiro de 2016

Elizeu Elieber Fachini

Um Sistema de Monitoramento para Caracterização de Algoritmos Distribuídos

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação (PPGCCS) da Universidade Federal de São Carlos como parte dos requisitos exigidos para a obtenção do título de Mestre em Ciência da Computação. Área de concentração: Engenharia de Software e Redes de Computadores.

Universidade Federal de São Carlos – UFSCar

Centro de Ciências em Gestão e Tecnologia – CCGT

Programa de Pós-Graduação em Ciência da Computação – PPGCCS

Orientador: Prof. Dr. Gustavo Maciel Dias Vieira

Sorocaba, SP

Fevereiro de 2016

Fachini, Elizeu Elieber

Um Sistema de Monitoramento para Caracterização de Algoritmos Distribuídos / Elizeu Elieber Fachini. -- 2016.
89 f. : 30 cm.

Dissertação (mestrado)-Universidade Federal de São Carlos, campus Sorocaba, Sorocaba
Orientador: Gustavo Maciel Dias Vieira
Banca examinadora: Islene Calciolari Garcia, Fábio Luciano Verdi
Bibliografia

1. Monitoramento. 2. Sistemas distribuídos. 3. Algoritmos distribuídos. I. Orientador. II. Sorocaba-Universidade Federal de São Carlos. III. Título.

Ficha catalográfica elaborada pelo Programa de Geração Automática da Biblioteca campus Sorocaba (B-So).
DADOS FORNECIDOS PELO(A) AUTOR(A)

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências em Gestão e Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Elizeu Elieber Fachini, realizada em 04/02/2016:

Prof. Dr. Gustavo Maciel Dias Vieira
UFSCar

Profa. Dra. Islene Calciolari Garcia
UNICAMP

Prof. Dr. Fabio Luciano Verdi
UFSCar

Agradecimentos

Meus sinceros votos de agradecimentos,

A Deus maiormente por cumprir suas promessas em minha vida, por me dar saúde e força para superar as dificuldades;

Aos meus pais, pela minha educação, incentivo e apoio dado em meus estudos e durante toda minha vida;

Ao professor Gustavo, por me orientar e direcionar, pelo apoio e principalmente pela confiança depositada em mim;

Ao professor Mário, orientador na graduação, por incentivar em continuar meus estudos;

Ao meu amigo Luiz Henrique, por possibilitar essa oportunidade através de nossa parceria no trabalho de conclusão da graduação;

À minha noiva e futura esposa, pela confiança, amor e perseverança em me apoiar em meus sonhos;

Aos meus muitos amigos por demonstrar confiança em minha capacidade, contribuindo no ânimo em terminar mais esta etapa.

Resumo

O monitoramento é o ato de coletar informações referentes às características e estado dos recursos de interesse. Ele pode ser utilizado para gerência e alocação de recursos, detecção e correção de falhas e também para avaliação de parâmetros de desempenho. Para realizar o monitoramento de modo automático é necessário a utilização de ferramentas, que tem funcionalidades referentes a captação, processamento, distribuição e apresentação dos eventos de monitoramento. Neste trabalho temos interesse em um sistema de monitoramento para dar suporte à execução experimental de algoritmos distribuídos, com o objetivo de relacionar o estado dos dispositivos com os dados da execução e, desta forma, permitir uma análise do uso de recursos do aglomerado pela aplicação. É necessário então uma ferramenta com características particulares como fazer a coleta de informações com pequeno intervalo de tempo, com baixa intrusividade e realizar o armazenamento total dos dados. Como não foi possível encontrar na literatura uma ferramenta com as características desejadas, desenvolvemos uma nova ferramenta de monitoramento chamada MSPlus. As características dessa nova ferramenta foram analisadas através de experimentos de forma isolada e em comparação a outra ferramenta. Adicionalmente, aplicamos a ferramenta em um sistema distribuído monitorando um algoritmo distribuído.

Palavras-chaves: Monitoramento. Sistemas distribuídos. Algoritmos distribuídos.

Abstract

Monitoring is the act of collecting information concerning the characteristics and status of resources of interest. It can be used to the management and allocation of resources, detection and correction of failures and also to the evaluation of performance parameters. To automatically accomplish the monitoring a tool is needed that has functionalities related the acquiring, processing, distributing and presenting of monitoring events. In this work we are interested in a monitoring system to give support to the experimental execution of distributed algorithms, with the objective of correlating the device status with the execution data and, this way, make possible an analysis of cluster resources used by the application. Then, it's needed a tool with particular characteristics, such as the ability to collect data with a small time period, with low intrusiveness and making the full data available. As was not possible find in the literature a tool with the features required, we developed a new monitoring tool named MSPlus. The features of this tool were evaluated through experiments with the isolated tool and comparing it with other tool. Additionally, we apply the tool in a distributed system to monitor a distributed algorithm.

Key-words: Monitoring. Distributed Systems. Distributed Algorithms.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Interface de Programação de Aplicação
CPU	Unidade Central de Processamento
E/S	Entrada/Saída
GB	Gigabyte
GMA	Arquitetura de Monitoramento de Grade
HDD	Dispositivo de Disco Rígido
IP	Protocolo de Internet
kB	kilobyte
MB	Megabyte
op/s	Operações por segundo
RRA	Arquivo Round Robin
SOs	Sistemas Operacionais
TB	Terabyte
TCP	Protocolo de Controle de Transmissão
UDP	Protocolo de Datagrama de Usuário
XML	Linguagem de Marcação Extensiva

Sumário

Introdução

O *monitoramento* é o ato de coletar informações referentes às características e estado dos recursos de interesse (ZANIKOLAS; SAKELLARIOU, 2005). Em qualquer sistema computacional, seja ele composto por um único computador até grandes aglomerados ou grades computacionais, é necessário obter dados sobre o estado de variados recursos de interesse para verificar e acompanhar o seu comportamento. O monitoramento pode ser utilizado para gerência e alocação de recursos, para detecção e correção de falhas e também para avaliação de parâmetros de desempenho (ZANIKOLAS; SAKELLARIOU, 2005). Isto é feito através de um conjunto de ferramentas de monitoramento, as quais captam informações desses recursos, as processam e as entregam aos usuários.

Nos sistemas operacionais existem diversos recursos que possibilitam capturar os estados desejados. No Linux ou Unix, os estados de um dispositivo ficam disponíveis através de arquivos em alguns diretórios, podendo ser facilmente acessados através da linha de comandos, enquanto que no Windows isto é feito indiretamente usando o conjunto de ferramentas de monitoramento PerfMon. O conjunto de ferramentas de monitoramento é também chamado de sistema de monitoramento. Esses sistemas facilitam a tarefa de coleta de informações, uma vez que automatizam várias tarefas repetitivas com uma precisão muito superior a qualquer medição manual. Se considerar um aglomerado ou grade computacional o problema é maior ainda, pois é necessário obter, transferir e agregar de forma coerente as informações de um conjunto de máquinas distintas. Esse tipo de monitoramento, assim como outras funções elementares de gerência de aglomerados, é ainda pouco oferecido entre os serviços providos pelos principais SOs. Esses serviços são usualmente realizados por um ou mais componentes de *middleware* (SACERDOTI et al., 2003; SOTTILE; MINNICH, 2002; SMITH et al., 2001; GUNTER; TIERNEY, 2003), sendo o trabalho de integração dos mesmos nem sempre trivial.

Existem diversos cenários em que os sistemas de monitoramento são utilizados. O mais comum é para observar em tempo real o comportamento do sistema operacional junto a aplicações que estão sendo executadas, buscando ver o impacto geral sobre o sistema ou sobre algum recurso específico (TESSER, 2011; SOTTILE; MINNICH, 2002; MASSIE; CHUN; CULLER, 2004). Porém, o monitoramento também pode ser usado em outras circunstâncias, como por exemplo, fazer uma análise *post-mortem* de computações para identificar particularidades ou caracterizar as computações feitas. Este tipo de aplicação usualmente sai do escopo dessas ferramentas, sendo comum nesses casos o surgimento de novos sistemas de monitoramento.

Neste trabalho temos interesse em um sistema de monitoramento para dar suporte

à execução experimental de algoritmos distribuídos, com o objetivo de relacionar o estado dos dispositivos com os dados da execução e, desta forma, permitir uma análise do uso de recursos do aglomerado pela aplicação. Para isso, é essencial coletar o maior número possível de métricas como vazão de rede, carga de CPU, vazão de disco, etc.

Ao empregar as ferramentas de monitoramento de aglomerado para caracterizar o desempenho de algoritmos distribuídos em função de parâmetros operacionais, a simples coleta de dados não é suficiente. Especificamente, precisamos associar os dados de forma sistemática às execuções experimentais do algoritmo sendo estudado, fazendo a correlação temporal entre operações executadas pelo mesmo e o estado do aglomerado.

Por ser um caso específico, é necessário então uma ferramenta com características particulares como fazer a coleta de informações com pequeno intervalo de tempo, com baixa intrusividade e realizar o armazenamento total dos dados. Com esses objetivos em mente usamos como ponto de partida uma ferramenta de monitoramento para observar um ambiente de usuário, apenas para avaliar os possíveis problemas que ela poderia ter para caracterizar uma aplicação. Para isso diminuímos o intervalo de captura, o que acabou causando um aumento significativo do uso de recursos do ambiente. Essa situação é inaceitável para o nosso propósito, pois o uso de recursos pode afetar na observação do algoritmo distribuído sendo usado.

Como não foi possível encontrar na literatura uma ferramenta com as características desejadas, nesse trabalho foi desenvolvida uma nova ferramenta de monitoramento chamada MSPlus. O resultado que conseguimos tem características um tanto diferentes de outras ferramentas de monitoramento existentes. E assim, contribuímos com uma ferramenta capaz de fazer coleta de dados com resolução de segundos por longo período de tempo e com uso mínimo dos recursos do ambiente.

As características dessa nova ferramenta foram analisadas através de experimentos com o MSPlus isolado e em comparação à primeira ferramenta utilizada. Os resultados obtidos mostram além do uso mínimo de recursos, a capacidade de MSPlus de ter os resultados de monitoramento precisos, sendo capaz de identificar pequenos soluços durante os experimentos.

Adicionalmente, aplicamos a ferramenta em um sistema distribuído monitorando um algoritmo distribuído e foi possível perceber algumas características interessantes de comportamento da aplicação em relação ao ambiente de testes. Dentre as características observadas podemos listar a situação de um dos nós ter problemas de funcionamento durante a execução do algoritmo e também o desempenho do algoritmo quando utilizando sistemas de arquivos diferentes.

Este trabalho está organizado da seguinte forma. No Capítulo 1 apresentamos os aspectos teóricos de sistemas de monitoramento, além de uma breve análise sobre

algumas ferramentas existentes. No Capítulo 2 descrevemos a ferramenta desenvolvida. No Capítulo 3 apresentamos resultados de experimentos feitos para analisar a ferramenta desenvolvida. No Capítulo 4 apresentamos os resultados dos experimentos caracterizando o Treplica, um ambiente de programação para aplicações replicadas de alto-desempenho.

1 Sistemas de Monitoramento

Neste capítulo apresentamos conceitos de sistemas de monitoramento, classificamos vários sistemas considerando uma taxonomia, além de fazer uma breve descrição de algumas ferramentas que inicialmente consideramos interessantes para este trabalho. Concluímos apresentando os requisitos de um sistema de monitoramento para algoritmos distribuídos e como as ferramentas descritas atendem a esses requisitos.

1.1 Monitoramento de Sistemas Distribuídos

Um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente (TANENBAUM; STEEN, 2007). Atualmente, dois formatos populares de sistemas distribuídos são os aglomerados (*clusters*) e as grades (*grids*). Os aglomerados são homogêneos e são formados por computadores semelhantes e com os mesmos sistemas operacionais. As grades também são formadas por diversos computadores conectados, mas se diferenciam por terem heterogeneidade nos equipamentos. Uma grade não possui padronização e pode ser formada por computadores e sistemas operacionais de vários tipos.

O sistema de monitoramento em um sistema distribuído é o responsável por capturar, processar e fornecer dados referentes ao que está acontecendo em cada nó pertencente a este tipo de sistema. Esses dados podem ser: quantidade de memória usada, taxa de transferência de dados do disco rígido, porcentagem da capacidade de processamento utilizado, entre outras medidas que variam dependendo de qual sistema de monitoramento, sistema operacional e *hardware* estão sendo usados. O sistema de monitoramento é importante para a detecção de erros, escalonamento de ambientes, depuração e análise de desempenho de aplicações, possibilitando armazenar dados históricos com a finalidade de analisar o comportamento do sistema por um período de tempo.

O monitoramento de um sistema distribuído ocorre através da implementação de quatro funcionalidades básicas: geração, distribuição, processamento e apresentação dos dados (TESSER, 2011). A relação entre essas quatro funcionalidades é mostrada no diagrama da Figura 1, onde as setas representam fluxo de dados. É possível observar que um sistema pode ser configurado para primeiro fazer o processamento local e depois a distribuição, como pode também distribuir os dados primeiro e fazer o processamento posteriormente. Essas funcionalidades podem variar dependendo do sistema de monitoramento, sua configuração e do pedido do cliente, sendo possível implementá-las apenas parcialmente. Por exemplo, é possível ocorrer o processamento e a apresentação dos dados somente no nó onde estão sendo gerados, sem ocorrer a distribuição dos mesmos (TESSER,

2011).

Figura 1: Relação das funcionalidades em sistemas de monitoramento

A geração de dados está diretamente ligada à coleta de informações dos recursos de interesse. Essa coleta é realizada através de sensores, que podem ser classificados em sensores passivos e ativos. O primeiro é tipicamente instalado em sistemas operacionais e fornece medições específicas do mesmo. No segundo, as medições são estimativas utilizando *benchmarks* personalizados, tendo maior intrusividade que o primeiro (ZANIKOLAS; SAKELLARIOU, 2005).

O sistema de monitoramento comporta-se de diversas maneiras quanto a disponibilidade dos dados: *online*, *semi-online* e *post-mortem* (BALATON et al., 2001; TESSER, 2011). Em sistemas *online* e *semi-online* os dados são atualizados frequentemente. No sistema *online* é possível o usuário visualizar os dados do comportamento de um sistema distribuído em tempo de execução, pois os dados são captados e distribuídos automaticamente. Em um sistema *semi-online* os dados de monitoramento somente são captados ao serem solicitados pelo cliente. No caso de *post-mortem*, os dados são apresentados ao término do processo observado ou posteriormente a um período pré-definido para a realização do monitoramento (TESSER, 2011).

A transmissão de dados entre componentes de um sistema de monitoramento, é feita através de dois modelos: *pull* e *push* (TESSER, 2011). O modelo *pull* ocorre quando a transmissão de dados é solicitada pelo receptor, ou seja, sob demanda. No modelo

push, as transmissões de dados são feitas sem a necessidade de solicitação por parte do receptor. Considerando o comportamento dos sistemas distribuídos, um sistema *online* utiliza métodos *push*, pois a cada intervalo de tempo os dados são transmitidos sem a solicitação do receptor. Porém, existem sistemas *online* que usam o método *pull*, mesmo que parcialmente.

1.1.1 A Arquitetura de Monitoramento de Grade

As especificações da arquitetura de monitoramento de grade ou GMA, sigla em inglês de *Grid Monitoring Architecture* (SMITH et al., 2001), foram elaboradas pelo *Open Grid Forum* com propósitos de atender aos requisitos específicos de um sistema de monitoramento, visando a interoperabilidade entre elas (TIERNEY et al., 2002).

A GMA consiste de três componentes básicos: produtor, consumidor e serviço de diretório, conforme é possível visualizar na Figura 2 (TIERNEY et al., 2002). O produtor é o responsável por disponibilizar os dados de monitoramento e publicar sua existência no diretório. O consumidor procura pelos produtores disponíveis no diretório e recebe os dados de monitoramento disponíveis (TESSER, 2011; ZANIKOLAS; SAKELLARIOU, 2005). O serviço de diretório contém as informações dos produtores e consumidores disponíveis, necessários para estabelecer a conexão entre eles, assim como as especificações dos tipos de eventos que produzem ou consomem (SMITH et al., 2001). Existe também um quarto componente adicional, chamado republicador, que possui características tanto de produtor quanto de consumidor. O republicador tem a função de consumidor, ao acessar os eventos de diversos produtores, e também se comporta como produtor, disponibilizando os eventos para consumidores em um nível mais alto na hierarquia.

A GMA define três tipos de interações entre produtores e consumidores, que são *publish/subscribe*, *query/response* e notificação (TIERNEY et al., 2002). Essas interações fazem uso dos modelos de transmissão *pull* e *push*. *Publish/subscribe* e *query/response* usam o modelo *pull* e a notificação usa *push*. Em *publish/subscribe* a interação é feita em três fases que consistem em uma assinatura para um tipo especifico de evento, um fluxo de eventos entre produtor e consumidor e um encerramento da assinatura. A assinatura é feita apenas pelo consumidor, o fluxo de eventos é do produtor para o consumidor e o encerramento é feito tanto pelo produtor quanto pelo consumidor. *Query/response* consiste de uma interação iniciada por um consumidor que realiza uma consulta e recebe uma resposta do produtor, que pode ter um ou mais eventos. Por fim, na interação de notificação os eventos são enviados diretamente de um produtor para um consumidor, sem que o consumidor os solicite.

Figura 2: Arquitetura de monitoramento em grade (TIERNEY et al., 2002)

1.1.2 Taxonomia de Sistemas de Monitoramento de Grade

A taxonomia de sistemas de monitoramento de Grade proposta por Zanikolas e Sakellariou (ZANIKOLAS; SAKELLARIOU, 2005) é usada para classificar em categorias os sistemas de monitoramento existentes. A partir de características em comum é possível chegar a qual categoria um sistema de monitoramento pertence. A taxonomia de sistemas de monitoramento é organizada usando níveis de zero a três, no qual cada nível representa uma categoria:

Nível 0: Os sensores comunicam-se diretamente com os consumidores, ou seja, os consumidores recebem os dados diretamente dos sensores responsáveis pela geração dos eventos. Nesse nível não existe a API de produtor.

Nível 1: Nesse nível os sensores podem ser implementados separadamente do produtor, mas ambos estão na mesma máquina e o produtor define a forma padrão como os sensores são acessados. Todos os eventos gerados pelos sensores são disponibilizados para acesso remotamente apenas pela API de produtor.

Nível 2: Nesse nível, além dos produtores, é disponibilizado também o uso de pelo menos um tipo de republicador. Republicadores de funcionalidades distintas podem ser empilhados, desde que seja de forma pré-definida.

Nível 3: Esse nível representa os modelos hierárquicos reconfiguráveis, que são sistemas de monitoramento altamente flexíveis que possibilitam que os produtores e republicadores sejam organizados arbitrariamente.

A taxonomia também inclui alguns qualificadores, os quais podem ser representados desta forma: L{0, 1, 2a, 2b, 2c, 3}.{H, N, A, V, G}.[S]. O primeiro qualificador representa o nível. O Nível 2 usa uma letra minúscula (a, b, c) para representar se o republicador é centralizado (a), se é simplesmente distribuído (b) ou se é distribuído com suporte a replicação (c). O segundo qualificador representa qual o tipo de entidade principal é monitorada, sendo máquinas (H de *Host*), redes (N de *Network*), aplicações (A), disponibilidade (V de *aVailability*) e sistemas genéricos (G). O último qualificador S é opcional e serve para identificar se o sistema é empilhável (S de *Stackable*). Considerando os qualificadores apresentados, um sistema L2c.H denota um sistema de segundo nível o qual tem como preocupação principal o monitoramento de máquinas e tem pelo menos um republicador que suporta replicação de dados.

1.2 Sistemas de Monitoramento

Existem diversos sistemas de monitoramento disponíveis para variados modos de uso. É possível também a criação e montagem de um sistema de monitoramento através do uso de várias ferramentas disponíveis, onde cada ferramenta faz apenas parcialmente as funções de um sistema de monitoramento. Nessa abordagem uma ferramenta é responsável pela captura dos dados, outra processa os dados e outra permite visualizar os dados processados.

Com base na taxonomia apresentada na seção anterior construímos a seguinte lista de sistemas de monitoramento existentes na literatura. Algumas dessas classificações podem ser encontradas com maiores detalhes em (ZANIKOLAS; SAKELLARIOU, 2005).

Autopilot (RIBLER et al., 1998) (L1.A): é um sistema de Nível 1 e preocupa-se principalmente em monitorar aplicações.

Ganglia (MASSIE; CHUN; CULLER, 2004) (L3.G): é um sistema de Nível 3 com monitoramento genérico. Esse sistema de monitoramento está descrito em maiores detalhes na Seção 1.2.2

Globus MDS (CZAJKOWSKI et al., 2001) (L3.G.S): é um sistema de Nível 3 com monitoramento genérico e empilhável.

GRID ICE (ANDREOZZI et al., 2005) (L0.G.S): é um sistema de Nível 0, genérico e empilhável.

GridRM (BAKER; SMITH, 2003) (L2a.H.S): é um sistema de Nível 2 com republicador centralizado, preocupa-se principalmente em monitorar rede e é um sistema empilhável.

GRM (BALATON et al., 2001) (L1.A.S): é um sistema de Nível 1, preocupa-se principalmente em monitorar aplicações e é empilhável. Esse sistema de monitoramento está descrito em maiores detalhes na Seção 1.2.3

HBM (STELLING et al., 1999) (L2b.V): é um sistema de Nível 2 com republicador distribuído, preocupado com o monitoramento de disponibilidade.

JAMM (TIERNEY et al., 2001) (L2b.G): é um sistema de Nível 2 com republicador distribuído e que tem preocupações genéricas de monitoramento.

MapCenter (BONNASSIEUX; HARAKALY; PRIMET, 2002) (L0.V.S): é um sistema de Nível 0, preocupa-se em monitorar a disponibilidade e é um sistema empilhável.

Mercury (BALATON; GOMBÁS, 2003) (L2b.G): é um sistema de Nível 2 com republicador distribuído e seu monitoramento é genérico.

MonALISA (LEGRAND et al., 2003) (L3.G.S): é um sistema de Nível 3 com monitoramento genérico e empilhável.

NetLogger (GUNTER; TIERNEY, 2003) (L2a.A): é um sistema de Nível 2 com republicador centralizado e tem como preocupação o monitoramento de aplicações. Esse sistema de monitoramento está descrito em maiores detalhes na Seção 1.2.4.

NWS (WOLSKI; SPRING; HAYES, 1999) (L2c.N): é um sistema de Nível 2 com republicador distribuído com suporte a replicação, destinado ao monitoramento de redes.

OCM-G (BALIŚ et al., 2004) (L2b.A): é um sistema de monitoramento de Nível 2 com republicador distribuído, que se preocupa com o monitoramento de aplicações.

Paradyn/MRNet (MILLER et al., 1995) (L3.A): é um sistema de Nível 3 com ênfase em monitoramento de aplicações.

Remos (DINDA et al., 2001) (L2b.N.S): é um sistema de monitoramento de Nível 2 com republicador distribuído, ênfase em monitoramento de rede e é um sistema empilhável.

R-GMA (COOKE et al., 2003) (L3.G.S): também é um sistema de Nível 3 com monitoramento genérico e empilhável.

SCALEA-G (TRUONG; FAHRINGER, 2004) (L2b.G): é um sistema de monitoramento de Nível 2 com republicador distribuído e suas características de monitoramento são genéricas.

Supermon (SOTTILE; MINNICH, 2002) (L3.G) : é também um sistema de Nível 3 com monitoramento genérico. Esse sistema de monitoramento está descrito em maiores detalhes na Seção 1.2.5.

Com a existência de uma grande quantidade de sistemas de monitoramento, é importante aprofundar sobre alguns destes. Sendo assim, descrevemos nas Subseções 1.2.2, 1.2.3, 1.2.4 1.2.5 e 1.2.6 as respectivas ferramentas: Ganglia (SACERDOTI et al., 2003), GRM (BALATON et al., 2001), Munin (EDGEWALL SOFTWARE, 2002), NetLogger (TIERNEY et al., 1998) e Supermon (SOTTILE; MINNICH, 2002). Estas ferramentas tem algumas características interessantes que as levaram a ser analisadas com maior atenção e descritas neste trabalho.

O Ganglia é um dos sistemas de monitoramento mais conhecidos e citados em projetos que utilizam sistemas de monitoramento em sistemas distribuídos. Além de armazenar os dados coletados, o Ganglia também possibilita utilizar apenas seu produtor, o Gmond. O Supermon é semelhante ao Ganglia, mas com menor intrusividade, já que os dados coletados não são propagados a todos os nós monitorados. O NetLogger oferece a possibilidade de analisar uma aplicação de um modo mais profundo. Caso haja suspeita de existir algum ponto crítico que cause gargalo de desempenho na aplicação analisada, o desenvolvedor chama a API do Netlogger que capta os estados dos dispositivos no sistema. O GRM é o que mais aproxima às necessidades deste projeto. Ele foi projetado para monitorar aplicações, tem flexibilidade de funcionamento e capacidade de se comportar em modo *post-morten*.

1.2.1 Características Analisadas

As ferramentas contém diversas características. Para descrever as ferramentas nas próximas subseções, usamos algumas características básicas propostas por Tesser (TESSER, 2011) que são:

Objetivo Principal: são as tarefas para os quais se pretende utilizar uma ferramenta.

Estrutura de distribuição de informações: é a maneira de como são distribuídas as informações de dados *push* ou *pull*, como descrito na Seção 1.1.

Armazenamento e manutenção das informações: é a definição se há ou não o armazenamento de dados e, caso exista, como é feita a manutenção dos mesmos e qual é o seu tempo de vida.

Possibilidade de integração com outras ferramentas: como as ferramentas de monitoramento são compostas por diversas funcionalidades pode ser interessante integrá-las, combinando as suas partes.

Apresentação das informações: as informações de monitoramento podem variar de acordo com o sistema. Existem sistemas que apresentam os dados graficamente ou possuem APIs que possibilitam o uso dos dados por outras ferramentas. Além da apresentação, essa característica se preocupa com a possibilidade de pós-processamento dos dados de acordo com a visualização desejada.

Considerando estas características em nosso trabalho. Temos o interesse em um sistema de monitoramento para dar suporte à execução experimental de algoritmos distribuídos, com o objetivo de relacionar o estado dos dispositivos com os dados da execução. Sendo assim, é necessário uma ferramenta capaz de fazer a coleta de informações com pequeno intervalo de tempo, com baixa intrusividade e realizar o armazenamento total dos dados.

1.2.2 Ganglia

Ganglia (SACERDOTI et al., 2003) é um sistema de monitoramento distribuído escalável para sistemas de computação de alto desempenho, como aglomerados e grades. Ele é baseado em um projeto hierárquico dirigido a federação de aglomerados. Ele se baseia em protocolo *multicast listen/announce* para monitorar o estado dentro do aglomerado e em árvore de conexões ponto-a-ponto para integrar os dados gerados quando em diferentes aglomerados (MASSIE; CHUN; CULLER, 2004). O Ganglia é um sistema de monitoramento *online*, o qual captura os dados e os processa em tempo de execução podendo visualizá-los praticamente em tempo real.

O Ganglia é composto pelo produtor Ganglia *monitoring daemon* (gmond) e o consumidor Ganglia *meta daemon* (gmetad). Em cada nó de um aglomerado é executado o gmond, que captura os dados e os transmite a todos os outros nós através do protocolo *multicast*. Sendo assim, qualquer nó tem a informação de todos os outros nós que compõem o aglomerado. O gmetad compõe a árvore de conexões ponto-a-ponto, a qual integra os dado dos aglomerados federados. Além disso, é responsável por buscar periodicamente os dados gerados nos nós conforme é possível visualizar na Figura 3.

A distribuição de dados utiliza os métodos *pull* e *push*. O gmond reúne os dados e os transmite periodicamente para todos os outros nós utilizando o método *push*. Já o

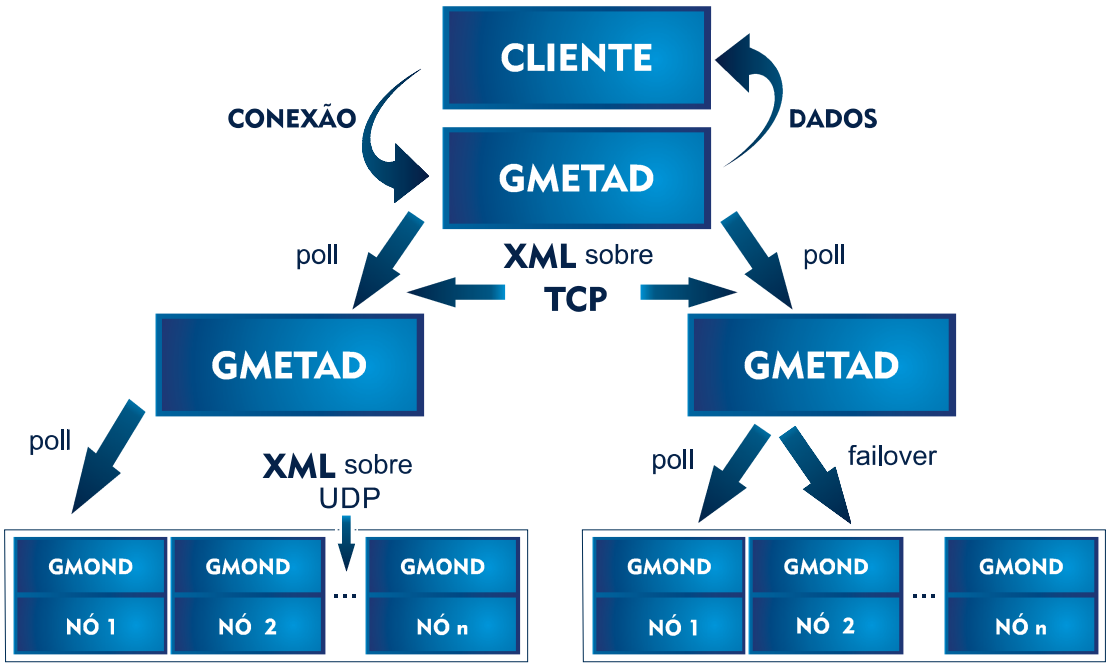

Figura 3: Arquitetura de Ganglia

gmetad reúne os dados periodicamente os buscando em algum nó, utilizando o método *pull*.

Para o armazenamento e visualização dos dados, o Ganglia utiliza a ferramenta RRDtool (*Round Robin Database*) que armazena e possibilita visualizar os dados em formato histórico. O RRDtool tem como característica básica fazer sumarização dos dados para diminuir o seu tamanho. Para as informações mais recentes é possível visualizá-las com maior detalhamento, enquanto as mais antigas apenas em dados sumarizados. Por exemplo, no último dia é possível visualizar os dados de hora em hora, mas na última semana só é possível visualizá-los de dia em dia.

1.2.3 GRM

GRM (BALATON et al., 2001) é um monitor *semi-online*, que recolhe informações sobre um aplicativo em execução em um sistema distribuído heterogêneo e fornece as informações coletadas para a ferramenta de visualização PROVE. GRM e PROVE fazem parte do P-Grid gráfico paralelo (ambiente de desenvolvimento de programas). Por usar a estratégia *semi-online* o usuário solicita a monitorização e algum tempo depois o monitor é capaz de obtê-los.

O GRM tem três partes essenciais, a biblioteca cliente, o monitor local e o monitor principal. Essas três partes principais são responsáveis pela geração, distribuição e armaze-

namento dos dados de eventos de monitoramento de um sistema distribuído. A biblioteca cliente é responsável pela geração dos eventos de rastreamento e estatísticas. Ela somente trabalha quando ocorre uma execução da aplicação monitorada e coloca os registros ou incrementos de contadores no *buffer* compartilhado com o monitor local. O monitor local executa em cada máquina e é responsável pelos eventos de rastreamento, armazenando os mesmos no *buffer* de memória compartilhada. Desta forma, caso o processo termine inesperadamente os dados estarão disponíveis ao utilizador até o momento da falha. O monitor principal coordena os monitores locais e coleta dados de rastreamento ao ser solicitado por um usuário. No caso do *buffer* encher, os dados são gravados em arquivo de texto em memória persistente. O monitor principal também realiza a sincronização de relógio.

Esse sistema pode ser configurado com algumas variações, uma delas seria ligar a biblioteca cliente diretamente ao monitor principal. Isto acarretaria em algumas limitações, entre elas a perda de dados caso ocorra uma falha em algum nó se o monitor principal não tiver guardado os dados. Além disto, essa configuração é menos flexível quanto as buscas dos dados, que deverão ser realizadas constantemente. A visualização dos eventos e estatísticas geradas é feita através da ferramenta PROVE. Essa ferramenta solicita os dados necessário para visualização ao monitor principal, que reúne todos os dados e retorna os valores para o PROVE.

1.2.4 Netlogger

O NetLogger (TIERNEY et al., 1998) foi projetado para realizar o monitoramento em condições reais de operação observando o comportamento de todos os elementos do caminho de comunicação aplicação-a-aplicação, a fim de determinar exatamente onde o tempo é gasto dentro de um sistema complexo. No NetLogger os componentes dos aplicativos são modificados para produzir registros protocolados de eventos de interesse em todos os pontos críticos do sistema distribuído. Os eventos de cada componente são correlacionados, o que permite caracterizar o desempenho de todos os aspectos do sistema e da rede em detalhes (GUNTER; TIERNEY, 2003). O NetLogger é um sistema de monitoramento *online*, o qual faz o processamento de todos os eventos e possibilita a visualização dos dados em tempo real.

No NetLogger a captura dos eventos é realizada por dois módulos, um para capturar os eventos da aplicação e o outro para capturar os eventos do comportamento do sistema e rede em geral, chamado de *wrappers*. No primeiro módulo a captura dos eventos na aplicação é feito através de uma API e uma biblioteca própria. Mas, para poder fazer esse monitoramento é necessário que o desenvolvedor do aplicativo chame a API NetLogger em cada ponto crítico do código para ligá-lo a biblioteca NetLogger. A captura de evento do restante do sistema é feita através da ferramenta *wrappers*, que é compatível

com vários sistemas Unix padrão e ferramentas de monitoramento de rede. Esse módulo é responsável por capturar essas informações e gerar eventos de monitoramento para o NetLogger. Os *wrappers* incluem o vmstat (CPU e monitoramento de memória), netstat (monitoramento de interface de rede), iostat (monitoramento de disco), e sunmget (acesso remoto a uma variedade de informações de monitoramento de rede).

Os dados gerados em NetLogger são armazenados de forma centralizada, todos os dados coletados são enviados a uma única máquina e porta, normalmente fora do sistema monitorado, utilizando o método *push*. O servidor executa um *daemon* servidor (netlogd), o qual recebe os eventos e os escreve em um disco local. Em caso de falha é necessário um servidor *backup*, até que se normalize o *daemon* principal. Por ser centralizado, não é possível o seu uso em grades de larga escala, onde centenas de máquinas podem fazer parte da execução de uma aplicação (SCHNORR, 2005). Para a análise e visualização é usado o NetLogger NLV, que possibilita a visualização dos dados em representação gráfica e interativa.

1.2.5 Supermon

Supermon (SOTTILE; MINNICH, 2002) é um flexível conjunto de ferramentas de alto desempenho e escalável para o monitoramento de aglomerados. O comportamento de um nó pode ser monitorado muito mais rápido do que com outros métodos. O Supermon usa um protocolo de transmissão de dados baseado em expressões simbólicas (S-expressões) em todos os seus níveis, de nós individuais até todo o aglomerado. O Supermon é dividido em três componentes distintos: um módulo do núcleo do Linux, um servidor de nó de dados (Mon) por máquina e um concentrador de dados (Supermon). O módulo do núcleo do Linux é responsável pela captura dos dados junto aos sensores e os disponibiliza em suas duas entradas no pseudo-sistema de arquivos sobre informações de processos (/proc). Em cada máquina o Mon atua como intermediário entre os dados disponibilizados pelo módulo do núcleo e os clientes TCP. Ele analisa os dados coletados, adiciona quantidade mínimas de informações, caso o cliente queira ter acesso aos dados de monitoramento de um único nó, o Mon lhe passa os dados sob demanda. Se o cliente quiser ver os dados de um conjunto de nós é usado o Supermon. Esse componente se conecta a um conjunto de nós que estão executando os servidores Mon, concentra os dados e os apresenta. O Supermon também tem a capacidade de servir como um republicador. Em um ambiente hierárquico o Supermon responsável por um aglomerado vira cliente de um Supermon de nível mais alto na hierarquia, que funciona como o concentrador de dados de vários aglomerados. Na Figura 4 é apresentado os principais componentes da arquitetura do Supermon e seus relacionamentos.

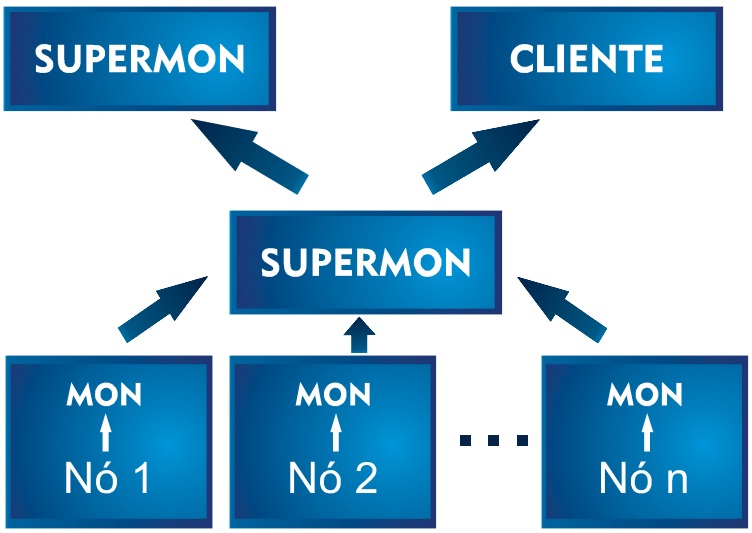

Figura 4: Arquitetura dos componentes do Supermon

1.2.6 Munin

Munin (EDGEWALL SOFTWARE, 2002) é capaz de monitorar diversos tipos de sistemas computacionais em tempo real e manter o histórico para eventuais análises futuras. Embora seja uma ferramenta de monitoramento conhecida, o Munin possui uma documentação muito superficial. Sendo assim, os poucos detalhes apresentados abaixo foram observados através do funcionamento da ferramenta, baseado nos conceitos de ferramentas de monitoramento existentes.

O Munin segue a arquitetura GMA apresentada na seção 1.1.1 e é composto por consumidor (munin-master), produtor (munin-node) e o sistema de diretório (arquivos de configuração). O munin-master tem as características de um sistema que usa o consumidor de forma centralizada, já que ele está presente em apenas um nó e gerencia o monitoramento fazendo solicitações aos produtores. Porém, vale ressaltar que o nó que o consumidor está nem sempre precisa ser um dos monitorados. Isto ocorre porque devido a existência do sistema de diretório o consumidor consegue solicitar os dados a cada produtor que está presente no diretório, indicado pelo seu endereço IP.

O método de distribuição utilizado pelo Munin é sempre *pull*. Isto ocorre porque quando o consumidor necessita dos dados, este os solicita aos produtores que respondem às solicitações. Porém, para que o produtor retorne os dados, este também os solicita a cada *plugin* de dispositivo instalado, que os retorna ao produtor só então os dados são retornados ao consumidor. Considerando que os produtores reúnem os dados a partir dos *plugins* instalados, o Munin é um sistema *plug-and-play*, sendo assim, basta a instalação dos *plugins* desejados para ser possível ao consumidor obter métricas com facilidade e

sem parar a execução.

Quanto a taxonomia de sistemas de monitoramento apresentada na Seção 1.1.2, o Munin pode ser considerado um sistema de monitoramento L2a.H. Esta taxonomia é de um sistema que contém um republicador de forma centralizada e realiza o monitoramento de nós em um sistema distribuído. O Munin tem um republicador, pois seu consumidor centralizado obtém os dados e os armazena, permitindo que outro consumidor possa acessar estes dados. Desta forma, o consumidor centralizado se comporta como um republicador.

Conforme foi citado, o Munin realiza o armazenamento dos dados mantendo um histórico do monitoramento. Para isto ele utiliza a ferramenta RRDtool (*Round Robin Database*) (OETIKER, 2012). No Munin o RRDtool é utilizado com suas funções de sumarização.

1.3 Análise das ferramentas

Considerando os objetivos deste trabalho o estudo de algoritmos distribuídos exige de um sistema de monitoramento algumas características particulares, como descrevemos na Seção 1.2.1. Como os sistemas descritos acima foram desenvolvidos para propósitos diferentes deste trabalho, suas características nem sempre atendem aos requisitos propostos. Na Tabela 1 são apresentados para cada ferramenta descrita acima o intervalo de coleta mínimo, o nível de consumo de recursos em uma execução de 1 segundo e se armazena totalmente os dados. No texto a seguir apresentamos uma breve análise que aponta os conflitos para cada ferramenta descrita na seção anterior.

Ferramenta	Intervalo	Consumo de recursos	Armazenamento total
Ganglia	60 s	Alto	Não
GRM	5 s	Baixo	Sim
Munin	60 s	Alto	Não
NetLogger	Sob demanda	Alto	Sim
Supermon	60 s	Alto	Não

Tabela 1: Características das ferramentas analisadas

O Ganglia é um sistema de monitoramento utilizado para auxiliar no gerenciamento e servir como histórico de uso de recursos dos ambientes em que está sendo executado. Sendo assim, o Ganglia coleta dados dos dispositivos de recursos no mínimo com intervalo de um minuto e sumariza os dados ao longo do tempo. O Ganglia também faz *broadcast* contínuo dos dados coletados. Como o intervalo de coleta é grande, o sistema sumariza os dados e cria tráfego de rede, o Ganglia foi descartado.

O Supermon tem propósitos semelhantes ao Ganglia, porém ele possui a vantagem que durante o monitoramento os nós não publicam seus estados a todos e não sumariza os dados. Infelizmente, o Supermon tem um intervalo de coleta de dados de um minuto. Sendo assim, não é possível utilizar esta ferramenta para os nossos objetivos.

O propósito de NetLogger é muito diferente de Ganglia e Supermon e pode auxiliar na identificação de pontos críticos como gargalos de desempenho dos ambientes monitorados. Porém, é preciso ter uma suspeita do ponto no código que está acontecendo algum gargalo de desempenho e inserir uma chamada da API do NetLogger neste ponto. Por essa necessidade de modificar o código das aplicações monitoradas o NetLogger foi descartado.

O GRM é o que mais chega perto de atender nossa necessidade, porque foi desenvolvido para caracterizar aplicações. Porém, por estar relacionado à ferramenta de desenvolvimento P-Grid, ele não pode ser usado por qualquer aplicação de aglomerado, sendo então descartado.

O Munin é uma ferramenta muito parecida com o Ganglia. Porém, ele permite utilizar com facilidade seus *plugins*. Sendo assim, usamos como ponto de partida o desenvolvimento de uma aplicação chamando apenas os *plugins* de interesse. No entanto, a frequência de amostragem necessária para ter granularidade fina é muito maior do que o padrão utilizado pelo Munin. Isto fez com que a intrusividade aumentasse substancialmente, tornando inviável a sua utilização. Na Seção 3.2 apresentamos os dados que representam este problema.

Desta forma, nenhuma das ferramentas estudadas atendeu nossas necessidades. Sendo assim, desenvolvemos uma nova ferramenta que está descrita no próximo capítulo.

2 MSPlus

2.1 Visão Geral

O MSPlus (Monitor System Plus) é uma nova ferramenta de monitoramento desenvolvida em Python para sistemas Linux que tem como principal intuito capturar e armazenar dados referentes aos estados dos dispositivos de um sistema computacional. A captura é feita através da leitura de arquivos no pseudo-sistema de arquivos sobre informações de processos (/proc), sendo esses arquivos mantidos abertos durante todo o monitoramento. O armazenamento dos dados é feito pelo RRDtool, ferramenta que é descrita na Seção 2.4.1.

O motivo do MSPlus ser apenas para sistemas Linux é porque esses sistemas são usados praticamente em todos os ambientes de execução de algoritmos distribuídos, como aglomerados e grades computacionais. Quanto a linguagem de programação, Python é uma linguagem de alto nível com bom desempenho de programação e execução, amplamente usada em ambientes de programação de alto desempenho e disponibilidade.

Considerando as características de um sistema de monitoramento, a ferramenta é composta por um consumidor e *plugins* de dispositivos que fazem parte da aplicação desenvolvida, mais o RRDtool. Além de fazer o monitoramento, o MSPlus permite gerar gráficos e exportar os dados em formato de texto.

Abaixo apresentamos conceitos como a arquitetura e taxonomia, as funcionalidades e detalhamento de implementação do MSPlus, além de um resumo do RRDtool.

2.1.1 Arquitetura e taxonomia

Considerando as definições de GMA apresentadas na Seção 1.1.1, um sistema de monitoramento de grade pode ser composto por produtor, consumidor e um serviço de diretório. O MSPlus não tem o produtor, pois os dados não ficam acessíveis durante a execução do monitoramento, devendo os mesmos serem coletados *post-mortem*. O MSPlus também não tem o arquivo de diretório, pois a função do mesmo é manter os dados sobre o produtor e consumidor, de forma que ambos possam ter um modo de se contactar. Sem um produtor, o serviço de diretório não é necessário. Por não ter em sua arquitetura o produtor e considerando a taxonomia apresentada na Seção 1.1.2, é possível com facilidade definir o nível de taxonomia do MSPlus. Como o sistema tem uma comunicação direta entre *plugins* e consumidor e o monitoramento é feito para observar os dispositivos de máquinas, a taxonomia do MSPlus é L0.H.

Não ter produtor mostra uma propriedade interessante do MSPlus, pois ele monitora sistemas distribuídos sem a distribuição dos dados. Isto mantém os dados de monitoramento independentes para cada nó, além de reduzir a intrusividade na banda de rede. A redução da intrusividade é importante, pois mesmo que a banda de rede em um sistema de aglomerados seja de alta velocidade, qualquer dado que o monitoramento trocar entre um nó e outro estará sendo contabilizado nos dados de monitoramento, comprometendo o resultado real da medida de desempenho deste dispositivo.

Desta forma, para realizar o monitoramento dos dados, cada nó fica completamente independente do ponto de vista do MSPlus. A ferramenta é iniciada em todos os nós monitorados por um *script* único, que também é responsável pela inicialização da aplicação avaliada. Posteriormente os dados são coletados em cada nó e processados *post-mortem* de forma centralizada, possivelmente integrado ao processamento dos dados do experimento.

A Figura 5 mostra uma visão geral dos componentes da ferramenta. Mostrando um único componente ligado ao RRDtool e, dentro deste componente, a comunicação do consumidor e os *plugins*.

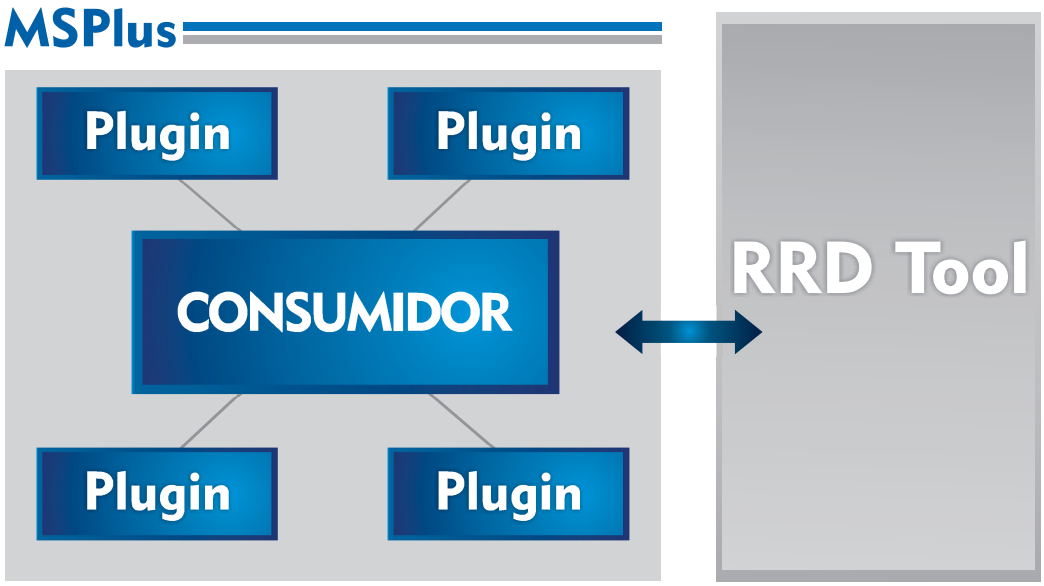

Figura 5: Visão genérica do MSPlus

2.1.2 Funcionalidades

A função básica do MSPlus é capturar os dados por um tempo dentro de um intervalo de um ou mais segundos, os enviando provisoriamente para a memória. Após

a captura de todas as amostras é feito um processamento prévio dos dados e os mesmos são armazenados no disco local, usando a biblioteca RRDtool. O processamento prévio é bem simples e consiste em apenas armazená-los, com exceção de algumas métricas que necessitam de tratamento pós captura.

Após o armazenamento dos dados eles devem ser coletados e todas as análises ocorrem *post-mortem*. Os gráficos servem para verificar os resultados do monitoramento de forma rápida e a exportação para ter os dados exatos de monitoramento, permitindo que eles sejam reprocessados e analisados posteriormente. Na criação de gráficos, o MSPlus usa a funcionalidade da função graph de RRDtool, passando os dados necessários. Na função de exportação, o MSPlus solicita os dados de um intervalo ao RRDtool e os exporta em arquivo de texto.

Os *plugins* de dispositivos tem uma tarefa importante para a captação dos dados, uma vez que cada um tem suas características próprias e agem de acordo com a forma que o dispositivo disponibiliza seus dados.

2.1.3 Uso da Ferramenta

Para executar o MSPlus no terminal Linux é necessário que seja feito a passagem de parâmetros e argumento, independente da função utilizada. Os parâmetros representam as funções, dados opcionais de monitoramento e os dispositivos. O argumento é o caminho do diretório onde serão armazenados os arquivos gerados pelas funções do MSPlus. Para função de monitoramento é importante o usuário criar o diretório especificamente para cada execução, pois como os arquivos gerados tem nomes iguais para cada dispositivo, caso execute o monitoramento mais de uma vez utilizando o mesmo diretório, os arquivos gerados irão sobrescrever os arquivos existentes no diretório.

O MSPlus aceita os seguintes parâmetros de execução:

`-r` ou `-run` Realiza a execução do monitoramento.

- **`-i <x>` ou `-interval <x>`** É o intervalo entre cada amostra de monitoramento, medido em segundos. Seu valor padrão é 1.
- **`-s <x>` ou `-sample <x>`** É a quantidade `x` de amostras que serão captadas durante o monitoramento, determinando em conjunto com o intervalo de amostragem o tempo total de execução do monitoramento. Exemplo, considerando que o intervalo seja 3 e a quantidade de amostras 500, o tempo de execução será de 1500 segundos. O valor padrão deste parâmetro é 100.
- **`-v <x>` ou `-save <x>`** É o número de amostras no qual a ferramenta realizará o processamento e armazenamento dos dados no disco. Como padrão o MSPlus

armazena os dados coletados na memória durante toda a execução do experimento, visando ter um melhor desempenho de disco. Porém, se a ferramenta for utilizada nos experimentos de longa duração, pode ter problemas com um consumo muito grande de memória. Este parâmetro possibilita a escolha de um intervalo para salvar os dados, liberando a memória utilizada. Como os dados são salvos apenas no final da execução, o valor padrão deste parâmetro é o mesmo da quantidade de amostras.

`-b <x>` ou `-heartbeat <x>` É o intervalo máximo aceitável entre amostras, medidos em segundos. Este visa diminuir os impactos de possíveis falhas que podem ocorrer durante o monitoramento. Por exemplo, se uma falha ocorre e seu tempo for inferior ao *heartbeat*, um valor médio é calculado e aplicado nas amostras desse intervalo. Se o intervalo entre as amostras com falhas for maior que a do *heartbeat*, neste caso todas as amostras desse intervalo ficam com valor '*unknown*'. O valor padrão deste parâmetro é 5.

`-g` ou `-graph` Realiza a geração de gráficos.

`-e` ou `-export` Exporta os dados armazenados para arquivos de texto.

Os parâmetros abaixo são usados em todas as funções para incluir os dispositivos de interesse. Exemplo: caso o usuário inclua apenas CPU, o monitoramento ou qualquer uma das funções (geração de gráficos ou exportação) serão executadas levando em conta apenas CPU.

`-c` ou `-cpu` Inclui CPU na execução das funções.

`-m` ou `-memory` Inclui memória na execução das funções.

`-n <if>,<if2>,...` ou `-network <if>,<if2>,...` Inclui a interface de rede `if` na execução das funções.

`-t <sda>,<sda2>,...` ou `-storage <sda>,<sda2>,...` Inclui o dispositivo de armazenamento `sda` na execução das funções.

Com os parâmetros possíveis e o argumento, uma linha de comando de MSPlus para monitorar memoria, CPU e os dispositivos de armazenamento **sda** e **sda2** por 300 segundos, com amostras no intervalo de 1 segundo e *heartbeat* de 5 segundos no terminal fica semelhante a:

```
$ python msplus.py -r -i 1 -s 300 -b 5 -m -c -s sda,sda2 /diretorio/
```

2.2 Monitoramento

Nesta seção são abordados detalhes da função de monitoramento, a principal função de MSPlus. Para facilitar o entendimento será usado além do texto, pseudocódigos em formato simples, exemplificando a implementação.

Ao executar a ferramenta passando o parâmetro de monitoramento é realizada uma verificação para analisar se foram passados parâmetros opcionais. Se alguns deles foram passados, a ferramenta verifica quais e utiliza os valores desses parâmetros. Caso não, são usados valores padrão. Com os valores definidos é chamado o método `Monitoring`.

O método `Monitoring` é o responsável por gerenciar as etapas de monitoramento que consistem em capturar (Algoritmo 1), processar (Algoritmo 2) e armazenar os dados (Algoritmos 3, 4 e 5). Basicamente, o método recebe os valores dos parâmetros, cria uma lista para armazenar os dados, realiza a captura dos dados e os armazena na lista criada inicialmente, que fica na memória durante o monitoramento. Estes dados são processados, retornando ao final a lista pronta para o armazenamento que é feito através do RRDtool.

A lista que armazena os dados durante o monitoramento ao ser criada contém apenas os campos relacionados à quantidade de dispositivos monitorados. Porém, quando a captura é iniciada, essa lista recebe diversas outras listas e ao final tem características de um banco de dados com quatro dimensões. Na primeira dimensão os dados são referentes aos dispositivos monitorados, na segunda são armazenados o tempo absoluto das amostras e na quarta os dados são os valores das métricas captadas. A terceira dimensão existe para todas, mas apenas realmente é usada quando é monitorado mais de um dispositivo do mesmo tipo, como é o caso de interfaces de rede ou dispositivos de armazenamento.

Após a criação da lista de dados, tem início a captura dos dados de monitoramento. Conforme pode-se ver no Algoritmo 1, a captura é feita através de dois laços encaixados. O primeiro laço a cada iteração faz uma comparação do tempo atual e o tempo da próxima amostra a ser captada. Se o tempo atual é menor do que o da próxima captura, verifica o tempo que falta e espera. Ao terminar este tempo de espera a aplicação entra no segundo laço pedindo os dados de monitoramento dos *plugins*, estes retornam os dados através de listas. Ao terminar o segundo laço, é atualizado o tempo da próxima captura e caso não chegou ao tempo final, o primeiro laço vai a próxima iteração.

O processamento dos dados apresentado no Algoritmo 2 manipula alguns dados a fim de criar novas métricas ou ajustar métricas captadas. Neste caso são utilizados também dois laços encaixados baseados na lista de dados. Basicamente, no primeiro laço a iteração passa pelos dados de um dispositivo e no segundo passa por cada amostra captada, chamando o método de processamento dos *plugins* monitorados. Após processar os dados para cada dispositivo ele retorna uma matriz que é guardada na lista contendo todas as métricas já processadas e prontas para o armazenamento.

Algorithm 1 Captura

```
1: início
2:     tempo_atual ← tempo_atual();
3:     prox_captura ← tempo_atual;
4:     tempo_final ← tempo_atual + (quant_amostras * intervalo) + 1;
5:     enquanto tempo_atual < tempo_final faça
6:         tempo_atual ← tempo_atual()
7:         tempo_espera ← prox_captura − tempo_atual;
8:         se tempo_espera > 0 então
9:             Espera tempo_espera;
10:        fim se
11:        y ← 0;
12:        enquanto y < quant_dispositivos faça
13:            listaDados[y] ← plugin_dispositivo[y]();
14:            y ← y + 1;
15:        fim enquanto
16:        prox_captura ← prox_captura + intervalo;
17:    fim enquanto
18: fim
```

Algorithm 2 Processamento

```
1: início
2:     x ← 0;
3:     enquanto x < comprimento(lista_dados) faça
4:         y ← 0;
5:         total_linhas ← comprimento(lista_dados[x]);
6:         enquanto y < total_linhas faça
7:             dado ← lista_dados[x][y];
8:             lista_dados ← Process(dado);
9:             y ← y + 1;
10:        fim enquanto
11:        x ← x + 1;
12:    fim enquanto
13: fim
```

Algorithm 3 Busca de nomes

```
1: início
2:     x ← 0;
3:     total_dispositivos ← comprimento(lista_dispositivo);
4:     enquanto x < total_dispositivos faça
5:         lista_nomes ← dispositivo[x].nomes();
6:         lista_tipos ← dispositivo[x].tipos();
7:         x ← x + 1;
8:     fim enquanto
9: fim
```

Antes de armazenar os dados no disco a aplicação realiza dois passos. O primeiro, apresentado no Algoritmo 3, é um laço simples e apenas tem o intuito de buscar os nomes e tipos de métricas. Os nomes são usados para identificar os arquivos de dados de cada métrica e os tipos são usados para definir o formato que os dados serão salvos pelo RRDtool nos arquivos. A criação dos arquivos é feita através de três laços encaixados, conforme mostra o Algoritmo 4. O primeiro laço varre os dispositivos distintos monitorados, o segundo laço varre os dispositivos iguais, quando há mais de um dispositivo de armazenamento ou interface de rede e o terceiro varre cada métrica. A cada iteração deste último é chamada uma função de RRDtool que cria cada arquivo no diretório definido.

Algorithm 4 Criação de arquivos de dados

```
1: início
2:     x ← 0;
3:     enquanto x < total_dispositivos faça
4:         y ← 0;
5:         quantidade_dispositivo ← total_dispositivo[x];
6:         enquanto y < quantidade_dispositivo faça
7:             z ← 0;
8:             enquanto z ← lista_nomes faça
9:                 parametros ← [nome, tipo];
10:                Chama RRDtool(parametros)
11:                z ← z + 1;
12:            fim enquanto
13:            y ← y + 1;
14:        fim enquanto
15:        x ← x + 1;
16:    fim enquanto
17: fim
```

A última parte do monitoramento consiste no armazenamento dos dados. O MSPlus chama o RRDtool passando os dados captados com o tempo absoluto da amostra e o RRDtool atualiza os arquivos de cada métrica. O armazenamento, representado no Algoritmo 5, é feito através de quatro laços. O primeiro varre cada dispositivo, o segundo varre o conjunto de amostras em um tempo absoluto, o terceiro varre as instâncias do mesmo dispositivo e o quarto laço varre as amostras de cada métrica chamando a função de atualização do RRDtool passando esses dados. O RRDtool adiciona os dados atualizando cada arquivo de métricas criado anteriormente.

Com os arquivos salvos, o método `Monitoring` termina a execução da aplicação. Considerando a arquitetura de monitoramento o método `Monitoring` realiza o papel de consumidor no MSPlus.

Algorithm 5 Armazenamento

```
1: início
2:     x ← 0;
3:     enquanto x < total_dispositivos faça
4:         y ← 0;
5:         enquanto y < total_amostras faça
6:             w ← 0;
7:             enquanto w < quantidade_dispositivo faça
8:                 z ← 0;
9:                 enquanto z ← lista_nomes faça
10:                    Cria dados;
11:                    RRDtool.update(lista_nomes[z], dados);
12:                    z ← z + 1;
13:                fim enquanto
14:                y ← y + 1;
15:            fim enquanto
16:            w ← w + 1;
17:        fim enquanto
18:        x ← x + 1;
19:    fim enquanto
20: fim
```

2.3 Plugins

No MSPlus os *plugins* tem a função de auxiliar o consumidor na captura dos dados e processamento das métricas de monitoramento dos dispositivos. Para este trabalho, os *plugins* foram desenvolvidos buscando o menor custo de recursos visando diminuir o impacto sobre os resultados dos experimentos.

Na versão inicial de MSPlus utilizamos os *plugins* de Munin. Estes *plugins* para capturar os dados chamam processos externos. Quando é executado na função para qual foi projetado esse procedimento causa um impacto pequeno, uma vez que o intervalo mínimo de captura do Munin é de um minuto. Mas como precisamos de um intervalo menor, adaptamos os *plugins* para capturar a cada segundo. Isto ocasionou um consumo de recursos alto e consequentemente causou uma intrusividade até maior que algumas aplicações. Como isto impactaria diretamente nos resultados do monitoramento, criamos os *plugins* baseando nas equações das métricas dos *plugins* do Munin. Porém, vale deixar claro que não utilizamos código do Munin, uma vez que o MSPlus é totalmente desenvolvido em Python e o Munin em Perl e Shell.

Tanto na captura dos dados quanto no processamento das métricas os *plugins* tem algumas particularidades dependendo do dispositivo monitorado. Porém, essas funções em cada um deles são simples. Como é possível ver no Algoritmo 6, que mostra a captura dos dados das métricas. Quando o *plugin* recebe uma solicitação do consumidor, ele abre o arquivo de dados do dispositivo, faz a leitura e responde a solicitação retornando os

valores dos dados captados ao consumidor. Vale ressaltar que os arquivos são abertos apenas na primeira vez para cada dispositivo monitorado e no início do monitoramento, após isto são feitas apenas leituras.

Algorithm 6 Execução dos Plugins

```
1: início
2:     se arquivo está aberto então
3:         lê arquivo;
4:         retorna leitura;
5:     senão
6:         abre arquivo;
7:         lê arquivo;
8:         retorna leitura;
9:     fim se
10: fim
```

Na etapa de processamento os *plugins* auxiliam, em alguns casos, com a manipulação dos dados gerando as métricas que não são possíveis de serem captadas diretamente nos arquivos de estado dos dispositivos. Diversas métricas do dispositivo de armazenamento e memória são geradas através de outras métricas. Além de gerar as métricas, o processamento as organiza em listas para o armazenamento posterior. Nas próximas seções abordamos um pouco mais sobre os *plugins* de cada dispositivo descrevendo a captura dos dados e o processamento das métricas, além de detalhar as métricas usadas no processamento e/ou disponibilizadas ao usuário.

2.3.1 Dispositivo de armazenamento

Os dispositivos de armazenamento são os componentes que tem o maior custo de utilização se comparado a outros componentes computacionais. Infelizmente, neste trabalho este é o *plugin* de dispositivo que requer maior cuidado em relação a intrusividade, pois durante a captura dos dados é o dispositivo que consome mais CPU. O *plugin* deste dispositivo pode ser usado pelo usuário para coletar amostras por dispositivo e/ou partições. Porém, é importante ressaltar que a captura dos dados é feita de forma independente por disco ou por partição.

Quando o *plugin* do dispositivo de armazenamento é chamado e inicia a captação de dados, ele busca um arquivo que pode variar dependendo do sistema Linux usado. Sendo assim, ele faz uma verificação sobre a existência dos arquivos de cada uma das partições ou dispositivo de armazenamento em `'/sys/block/%s/stat'`, onde `'%s'` é o nome do dispositivo de armazenamento. Se algum dos dispositivos ou partições não estivem nesse primeiro arquivo, é aberto o arquivo `'/proc/diskstats'`. Se o monitoramento estiver usando o segundo arquivo e monitorando mais que um dispositivo, é aberto um arquivo para cada um dos dispositivos de armazenamento monitorado.

O processamento feito pelo *plugin* deste dispositivo é essencial, pois todas suas métricas finais são geradas a partir do processamento dos dados captados durante o monitoramento. Para gerar as dados das métricas finais, é feito o processamento usando um laço para cada dispositivo de armazenamento monitorado. Dentro deste laço o processamento de cada dado de estado das métricas passa por três etapas. Em cada etapa são geradas métricas, porém o usuário vê apenas as métricas geradas pela segunda e terceira etapas.

Na primeira etapa são geradas métricas parciais a partir da diferença entre os valores do estado atual que está sendo processado e o valor do estado anterior, conforme apresentado na Equação 2.1.

$$valor\_metrica = (valor\_atual - valor\_anterior) \tag{2.1}$$

Estas métricas processadas são apenas utilizadas na geração de amostras das métricas finais. Segue abaixo um resumo do que é cada uma dessas métricas parciais:

Total de leituras *(read_ios):* é o número total de leituras realizadas com sucesso.

Total de escritas *(write_ios):* é o número total de escritas realizadas com sucesso.

Tempo médio de leitura *(rd_ticks):* é o tempo em milissegundos gasto com leitura.

Tempo médio de escrita *(wr_ticks):* é o tempo em milissegundos gasto com escrita.

Setores lidos *(rd_sectors):* é o número de setores lidos com sucesso.

Setores escritos *(wr_sectors):* é o número de setores escritos com sucesso.

Tempo médio total *(total_ticks):* é o tempo em milissegundos, gasto com leitura e escrita.

A segunda etapa consiste em gerar as métricas finais a partir das métricas já processadas inicialmente. Embora sejam métricas finais, algumas destas são usadas na geração de novas métricas na terceira etapa. As métricas obtidas nesta etapa são:

Leituras por segundo *(read_io_per_sec):* é o número médio de leituras realizadas com sucesso por segundo. É gerada através do resultado da divisão da métrica de leituras realizadas com sucesso pelo intervalo e tem sua estrutura conforme a Equação 2.2.

Escritas por segundo (*write_io_per_sec*): é o número médio de escritas realizadas com sucesso por segundo. É gerada através da métrica de escritas realizadas com sucesso dividida pelo intervalo, tendo estrutura conforme a Equação 2.2.

$$valor\_metrica\_por\_segundo = valor\_metrica/intervalo \tag{2.2}$$

Bytes lidos por segundo (*read_bytes_per_sec*): é o número médio de bytes lidos com sucesso por segundo. Como mostra a Equação 2.3 esta métrica é gerada através do resultado da divisão da métrica de leitura de setores pelo intervalo de captura, multiplicado pelo tamanho dos setores.

Bytes escritos por segundo (*write_bytes_per_sec*): é o número médio de bytes escritos com sucesso por segundo. Assim como o anterior, é gerada através do resultado da divisão da métrica de escrita de setores pelo intervalo de captura, multiplicado pelo tamanho dos setores. A Equação 2.3 representa a estrutura do processamento feito para obtenção dessa métrica.

$$valor\_metrica\_por\_segundo = (valor\_metrica/intervalo) * bytes\_por\_setor \tag{2.3}$$

Utilização (*utilization*): é a porcentagem de tempo em que o dispositivo esteve ocupado com leitura e escrita. É uma das métricas mais simples e é gerada a partir da métrica tempo médio total (*total_ticks*) dividida pelo intervalo de captura, tendo a mesma estrutura da equação 2.2. Um ponto importante é que nesta segunda etapa esta métrica representa o tempo em milissegundos que ficou ocupada. Somente depois na terceira etapa a métrica é dividida por dez, a convertendo para a porcentagem média de utilização com leitura e escrita em segundos. A divisão por dez é a simplificação da equação de conversão em porcentagem do intervalo em segundos dividido por mil, multiplicado por cem, no qual resulta uma divisão por dez.

Na terceira e última etapa são geradas o restante das métricas finais, estas junto as da segunda etapa podem ser visualizadas pelo usuário ao final do monitoramento. As métricas geradas nesta etapa tem maior complexidade, além de algumas particularidades em sua geração. Elas são obtidas a partir de uma ou mais das métricas, como descrito a seguir:

Duração média de E/S (*svctm*): é o tempo médio de ida e volta da solicitação de disco não incluindo tempo de fila. É gerada a partir do tempo de utilização (*utilization*) dividido pela soma de leituras e escritas por segundo (*read_io_per_sec* e

write_io_per_sec), após isso é dividida por mil, convertendo o resultado de milissegundos para segundos, conforme é possível ver na Equação 2.4. Vale ressaltar que caso a soma do total em segundos do tempo de leituras e escritas realizadas com sucesso for zero, o resultado para a duração média de E/S também será zero.

$$svctm = utilization/(read\_io\_per\_sec + write\_io\_per\_sec)/1000 \tag{2.4}$$

Espera média (avgwait): é o tempo médio de espera de E/S por solicitação do início ao fim da operação, incluindo tudo inclusive a fila. Conforme a equação 2.5, esta métrica é gerada a partir da soma do tempo de leituras e escritas (*rd_ticks*, *wr_ticks*), dividido pelo número de operações de leituras e escritas (*total_ios*). Ao final, é divida por mil, convertendo o resultado de milissegundos para segundos. Vale ressaltar que caso o número de operações de leituras e escritas (*total_ios*) for zero, a espera média também é zero.

$$avgwait = (rd\_ticks + wr\_ticks)/total\_ios/1000 \tag{2.5}$$

Espera média de leitura (avgrdwait): é o tempo médio de espera para uma leitura, a partir da solicitação do inicio ao fim. É gerada a partir do tempo de leitura (*rd_ticks*), dividido pelo número de operações de leituras (*read_ios*) com uma divisão por mil, para converter o resultado em segundos. Ressaltando que caso o número de operações de leitura (*read_ios*) for zero, o resultado final da métrica também será zero. A Equação 2.6 apresenta a equação desta métrica.

$$avgrdwait = rd\_ticks/read\_ios/1000 \tag{2.6}$$

Espera média de escrita (avgwrwait): é o tempo médio de espera para uma escrita, a partir da solicitação do inicio ao fim. Conforme é apresentado na Equação 2.7, esta métrica é gerada a partir do tempo gasto em escritas (*wr_ticks*) dividido pelo número de operações de escrita (*write_ios*), e convertida em segundos com a divisão por mil. Se o número de operações de escrita (*write_ios*) for zero, o resultado para a métrica também é zero.

$$avgwrwait = wr\_ticks/write\_ios/1000 \tag{2.7}$$

Tamanho médio de leitura (avgrdrqsz): esta métrica mostra o tamanho médio das leituras em kilobytes. Conforme é apresentado na Equação 2.8, ela é obtida a partir do resultado da quantidade de leitura de setores realizadas com sucesso (*rd_sectors*) multiplicada pela quantidade de bytes por setor (*bytes_po_setor*), dividido por mil

e depois dividido pelo número total de leituras feitas com sucesso (*read_ios*). Caso o número total de leituras for zero, o tamanho médio de leitura também será zero.

$$avgrdrqsz = rd\_sectors * bytes\_por\_setor/1000/read\_ios \quad (2.8)$$

Tamanho médio de escrita (avgwrrqsz): esta métrica tem as mesmas propriedades da anterior com a diferença de que mostra o tamanho médio de escritas realizadas durante o intervalo de coleta. Como é possível notar na Equação 2.9, ela é obtida a partir do resultado da quantidade de escrita de setores realizadas com sucesso (*wd_sectors*) multiplicada pela quantidade de bytes por setor (*bytes_por_setor*), dividido por mil e depois dividido pelo número total de escritas feitas com sucesso (*write_ios*). Caso o número total de escritas for zero, o tamanho médio de escrita também será zero.

$$avgwrrqsz = wr\_sectors * bytes\_por\_setor/1000/write\_ios \quad (2.9)$$

2.3.2 CPU

O *plugin* de CPU é bem simples se comparado ao de armazenamento. Ele não tem pré-verificações e os dados referentes ao estado de CPU estão sempre no arquivo '`/proc/stat`'. O processo de captura é representado pelo Algoritmo 6. A unidade de medida dos dados captados para este dispositivo é a porcentagem de utilização e o valor de cada métrica é a soma do estado da métrica em cada núcleo. Supondo que a utilização de uma CPU de quatro núcleos está em 100% cada, o valor do estado da métrica é igual a 400%.

As métricas de CPU disponibilizadas ao usuário não sofrem alteração no processamento. Porém, dependendo da versão do núcleo do Linux, algumas dessas métricas não podem estar disponíveis, não aparecendo na lista de resultados do monitoramento. Abaixo, segue uma breve descrição das métricas de CPU que o *plugin* disponibiliza aos usuários:

Processos de usuário (cpu_user): é a porcentagem de CPU utilizada por processos comuns executando em modo usuário.

Processos normais de usuário(cpu_nice): é a porcentagem de CPU utilizada por processos de baixa prioridade executando em modo usuário.

Processos de sistema (cpu_system): é a porcentagem de CPU utilizada por processos executando em modo supervisor.

Sem uso (cpu_idle): é a porcentagem de CPU que não está sendo utilizada.

Processos esperando (cpu_iowait): é a porcentagem de CPU utilizada por processos que estão esperando alguma iteração de entrada ou saída para terminar.

Interruções (cpu_irq): é a porcentagem de CPU utilizada por parte de interrupções de serviços.

Interrupções de núcleo (cpu_softirq): é a porcentagem de CPU utilizada por interrupções de serviços do núcleo Linux.

Espera involuntária (cpu_steal): é a porcentagem de CPU utilizada por espera involuntária.

Processos convidados (cpu_guest): é a porcentagem de CPU utilizada por execuções de processos convidados.

2.3.3 Memória

O *plugin* de Memória é tão simples quanto o de CPU, os seus dados de estado estão no arquivo `'/proc/meminfo'`. Porém, vale ressaltar que no caso de memória as métricas apresentadas ao final da execução do monitoramento são apenas algumas das métricas que estão disponíveis no arquivo de dados deste dispositivo. Como baseamos os *plugins* no Munin e este tem as métricas de nosso interesse, não vimos necessidade em incluir ou eliminar alguma métrica.

Para memória o processamento cria as métricas *apps* e *swap*, que são geradas a partir de outras que são usadas da maneira que foram captadas. Algumas das métricas captadas são descartadas no fim do processamento. Abaixo segue uma breve descrição das métricas utilizadas no processamento e disponibilizadas ao usuário. Informações sobre todas as métricas de memórias podem ser encontradas em (SMITH, 2007).

Cache do núcleo (slab): é a quantidade total de memória, em kilobytes, usada pelo núcleo para armazenar em cache estruturas de dados para seu próprio uso.

Tabela de paginação (page): é a quantidade total de memória, em kilobytes, dedicada para tabelas de paginação.

Espaço de endereço virtual (vmalloc): é a quantidade total de memória, em kilobytes, usada para o espaço de endereço virtual.

Aplicações (apps): é a quantidade total de memória, em kilobytes, que está sendo usada por aplicações. Como é possível ver na Equação 2.10, esta métrica é gerada

a partir da quantidade total de memória menos a quantidade de memória que está sendo usada em algum tipo de cache, buffer, página ou que esteja livre.

$$apps = total - free - buffers - cached - slab - page - swapc \tag{2.10}$$

Livre (free): é a quantidade de memória RAM física, em kilobytes, livre.

Buffers (buffers): é a quantidade de memória RAM física, em kilobytes, usada para buffers de arquivos.

Cache (cached): é a quantidade de memória RAM física, em kilobytes, usada para memoria cache.

Área de troca de cache (swapc): é a quantidade da área de troca *swap*, em kilobytes, usada para memoria cache.

Área de troca (swap): é a quantidade total de memória, em kilobytes, que está sendo usada para a área de troca *swap*. Esta métrica é gerada, conforme é possível ver na Equação 2.11 a partir da quantidade total disponível para troca subtraindo a quantidade livre disponível para troca.

$$swap = swaptotal - swapfree \tag{2.11}$$

Estimativa de uso (committed): é a quantidade total de memória, em kilobytes, estimada para completar a carga de trabalho.

Mapeamento (mapped): é a quantidade total de memória, em kilobytes, que está sendo utilizados para mapear os dispositivos, arquivos ou bibliotecas utilizando o comando de mapeamento de memória.

Ativo (active): é a quantidade total de *buffer* ou memória cache, em kilobytes, que está em uso ativo.

Inativo (inactive): é a quantidade total de *buffer* ou memória cache, em kilobytes, que está livre e disponível.

2.3.4 Rede

O *plugin* de rede é o que tem o menor número de métricas por dispositivo, a vazão de bytes por segundo recebidos (download) e enviados (upload). Porém, é possível realizar o monitoramento de mais de um dispositivo de rede, chamados de interfaces. O arquivo de dados é o `'/proc/net/dev'` e para cada interface é aberto um arquivo e lido de forma independente.

O dispositivo de rede, entre os quatro dispositivos monitorados, é o mais simples. O processamento desse *plugin* consiste apenas em adequar os dados para o armazenamento, sendo assim ele apenas retorna os valores de recebidos e enviados para cada interface monitorada.

`Dados recebidos (Download):` é a quantidade total, em bytes, de dados recebidos pela interface de rede

`Dados enviados (Upload):` é a quantidade total, em bytes, de dados enviados pela interface de rede.

2.4 Armazenamento dos Dados

Como descrito anteriormente, o MSPlus usa uma outra ferramenta para armazenar seus dados de monitoramento, o RRDtool. A escolha de utilizar uma ferramenta para armazenar os dados está justamente na facilidade que a ferramenta proporciona, além de processar automaticamente alguns dados coletados, utiliza um formato binário compacto, além de ter algumas funções como a criação dos gráficos. Abaixo está descrito esta ferramenta.

2.4.1 RRDtool

RRDtool (OETIKER, 2012) é um banco de dados *round robin*, *open source*, capaz de armazenar registros de dados de alta performance e um sistema de gráficos para dados de séries temporais. Segundo (OETIKER, 2012), o RRDtool permite ao usuário registrar e analisar dados de qualquer tipo de fonte de dados, de forma eficiente e sistemática, além de ser uma ferramenta que pode ser facilmente utilizada e integrada a softwares de diversas linguagens como Shell script, Perl, Python, Ruby, Lua e TCL.

O RRDtool é um banco de dados *round robin* por utilizar no armazenamento de dados o algoritmo de revezamento circular. Este algoritmo é um algoritmo utilizado normalmente para o escalonamento de CPU. Porém, aplicado a banco de dados possibilita ao RRDtool a capacidade de ter um tamanho fixo para o armazenamento de dados. A razão disso é porque o algoritmo de revezamento circular trabalha com uma quantidade fixa de dados e um ponteiro. Para exemplificar, o algoritmo de revezamento circular é semelhante a um relógio analógico, onde os dados do RRDtool são os segundos e o ponteiro de segundos do relógio aponta para qual dado está atualmente ativo. Cada vez que o relógio passa um segundo, equivale a escrita de um dado e o ponteiro passa ao próximo dado. Quando o relógio passa por todos os segundos de um minuto, ele volta a passar novamente pelos mesmos segundos. Da mesma forma o ponteiro do RRDtool quando passa por todos os dados, volta ao início e os dados mais antigos vão sendo

sobrescritos. Isto mantém o tamanho da base de dados sempre constante desde a sua criação. Algo importante a ressaltar é que o ponteiro somente vai ao próximo registro quando a escrita é consolidada.

O RRDtool ao ser criado também define o intervalo de tempo para a escrita de cada registro. Isto possibilita ao usuário enviar diversas amostras de dados dentro do mesmo intervalo, atividade esta chamada de aquisição de dados (*data acquisition*). Quando o intervalo termina, o RRDtool realiza a escrita do registro para o intervalo anterior, fazendo a consolidação do dado. Os dados podem ser consolidados de três formas: mínimo, máximo ou a média. Sendo assim, se definido que será usado a média, todos os dados enviados durante o intervalo são salvos em um registro temporário e ao final do intervalo é registrada a média de todos os dados.

Os dados de RRDtool são armazenados em arquivos RRD, que podem conter várias seções RRAs. Estas seções podem ter diversas configurações que são definidas ao criar o arquivo RRD. Cada RRA pode ter intervalo diferente para a entrada de dados. Normalmente, esses intervalos se remetem a medidas temporais como horas, dias, meses e anos. Com o passar do tempo os dados vão se sobrescrevendo a medida que atingem a quantidade máxima de dados para cada RRA. Um RRA de intervalo menor tende a atingir o limite em menor tempo que um RRA com intervalo maior. Desta forma, é que é feita a sumarização dos dados pelo RRDtool. Se o usuário souber quanto tempo será armazenado, é possível criar um arquivo com o tamanho exato de forma que a sumarização não aconteça.

A utilização dos dados armazenados no RRDtool podem acontecer de variados modos através de algumas funções como criação dos dados, exportação em extensão XML e visualização do banco de dados. No caso de MSPlus, essas funções foram usadas na geração de gráficos e exportação.

2.5 Gráficos

A função de criação de gráficos é bem simples comparada a função de monitoramento. Basicamente, ela apenas tem um laço na função, que varre os dispositivos para os quais o usuário deseja criar os gráficos. A cada iteração é chamado o método `Graph` das classes dos dispositivos. Esses métodos são responsáveis pela criação de uma lista de parâmetros e de chamar a função *Graph* de RRDtool.

No Algoritmo 7 é possível ver uma representação de forma genérica do método `Graph` do dispositivo de armazenamento, que é o método que tem a maior quantidade de laços. No primeiro laço ele varre a quantidade dispositivos de armazenamentos monitorados. No segundo laço como este dispositivo tem mais de um gráfico, ele passa pela criação de cada gráfico. E no terceiro laço é montado a cada interação uma lista com os

parâmetros e chamada a função *Graph* de RRDtool.

Algorithm 7 Função Gráfico

```
1: início
2:     x ← 0:;
3:     enquanto x < QuantidadeDispositivos faça:
4:         tempoInicial ← RRDtool.first(dispositivo.rrd);
5:         tempoFinal ← RRDtool.last(dispositivo.rrd);
6:         y ← 0:;
7:         enquanto y < QuantidadeGraficos faça
8:             z ← 0;
9:             enquanto z < metricas faça
10:                lê dadosGrafico;
11:            fim enquanto
12:            chama RRDtool.graph(tempoInicial, tempoFinal, dadosGrafico);
13:        fim enquanto
14:    fim enquanto
15: fim
```

A cada vez que a função *Graph* de RRDtool é chamada, o método `Graph` das classes dos dispositivos passam como argumento a lista de parâmetros. Essa lista de parâmetros nada mais é que uma lista com informações para a criação dos gráficos, como por exemplo o tamanho do gráfico, quais arquivos de armazenamento serão usadas, entre outras. A partir daí, o RRDtool cria os gráficos e os salva no mesmo diretório onde estão os arquivos de dados.

O MSPlus tem a possibilidade de criar 8 gráficos distintos. Três representam todas as métricas dos dispositivos de CPU, memória e interface de rede. Os demais representam os gráficos relacionados aos dispositivos de armazenamento que são: tamanho médio das operações de E/S, número médio de operações de E/S, latência, vazão do disco e utilização do disco.

2.6 Exportação de Dados

A função de exportar é um pouco mais complexa em relação a função de geração de gráficos. Porém, é importante citar que há algumas semelhanças entre ambas, pois os arquivos de texto exportados pela função *Export* são baseados nos dados usados nos gráficos. Os arquivos contém os dados separados por cada gráfico ao serem gerados, apenas para facilitar na geração de gráficos com outras ferramentas. No entanto, estes arquivos contém todas as métricas de MSPlus.

Como é apresentado no Algoritmo 8, há uma função que busca o tempo inicial e final do monitoramento através do RRDtool. Esta informação é necessária, pois para solicitar os dados RRDtool é preciso definir um intervalo passando o tempo inicial e final.

Para cada métrica é necessário uma solicitação. Sendo assim, as métricas são solicitadas e os dados retornados são armazenados no arquivo de texto.

Algorithm 8 Função Export

```
1: início
2:     x ← 0:;
3:     enquanto x < dispositivos faça:
4:         tempoInicial ← RRDtool.first();
5:         tempoFinal ← RRDtool.last();
6:         y ← 0;
7:         enquanto  façay < QuantidadeDispositivo:
8:             enquanto  façay < QuantidadeMetricasDispositivo:
9:                 dados ← RRDtool.graph(tempoInicial, tempoFinal);
10:            fim enquanto
11:            chama File.Write(dados);
12:        fim enquanto
13:    fim enquanto
14: fim
```

3 Validação Experimental

Nesse capítulo apresentamos os resultados de experimentos realizados com o objetivo de validar se o MSPlus funciona como projetado e se atende aos requisitos definidos para a ferramenta. Para esses testes focamos no comportamento de uma única máquina para enfatizar as características mais marcantes da ferramenta: a granularidade, o baixo consumo de recursos e a confiabilidade dos dados.

Todos os testes foram executados em uma máquina x86_64, com processador Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz, memória de 8GB e um HDD Sata de 1TB. Essa é uma máquina menos robusta do que usualmente encontrado em aglomerados, o que é interessante para mostrar que o MSPlus exige poucos recursos do sistema. Nesta máquina rodava o SO Ubuntu Linux 14.04.2 LTS, Python 2.7.4 e RRDtool 1.47.

3.1 Granularidade

Para usar o monitoramento na caracterização de uma aplicação é essencial saber o máximo possível do comportamento do sistema computacional afim de correlacioná-las com os dados da aplicação, para isso é necessário uma granularidade fina. Com a coleta das informações sendo feitas em intervalos pequenos é possível identificar o início de aplicações, mudanças de comportamento durante alguma ação e até anormalidades que ocorreram durante a execução da aplicação caracterizada.

Para ilustrar a granularidade do MSPlus realizamos um experimento de aproximadamente 3 h (11.000 s) realizado durante o uso normal da máquina. Com este experimento é possível mostrar se os dados estão sendo coletados na frequência desejada, além de observar algum comportamento interessante do sistema. Os resultados dos experimentos estão disponíveis nos gráficos das Figuras 6, 7 e 8 que representam CPU, memória e vazão de disco respectivamente.

A medida de CPU da máquina de teste ilustra como a granularidade de medida da métrica pode ser útil. Na Figura 6 (a) observamos uma anomalia por volta de 5000 s. Como possuímos todos os dados disponíveis com granularidade de 1 s, foi possível analisar com mais detalhes os valores da métrica ao redor desse instante de tempo como ilustrado nas Figuras 6 (b), (c) e (d). Assim, podemos observar que algum processo começou a ocupar 100% de um dos núcleos da CPU por volta de 4808 s e podemos usar esse valor para correlacionar o comportamento de outras métricas e também de dados de diagnóstico ou desempenho coletados por outras ferramentas.

Ainda na mesma execução podemos ver a Figura 7 (a) que mostra o consumo de

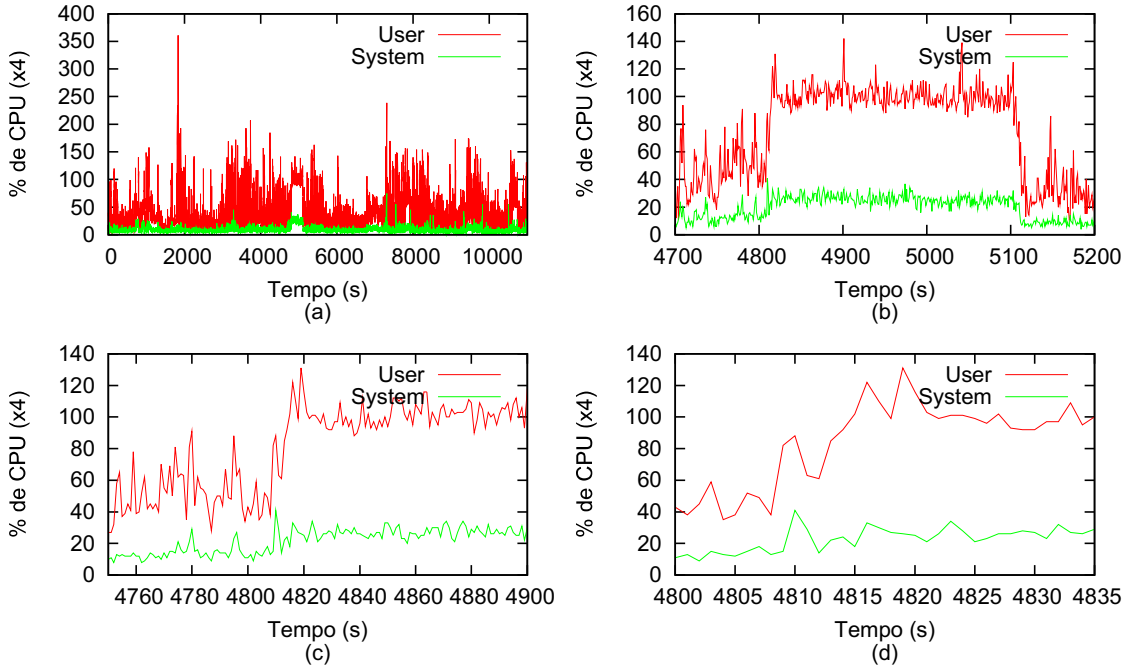

Figura 6: CPU

memória por vários componentes do sistema e Figura 7 (b) que mostra o volume de dados transmitidos pela rede. A Figura 8 mostra a vazão do dispositivo de E/S (disco), onde a Figura 8 (a) mostra essa vazão em operações de E/S e a Figura 8 (b) em megabytes.

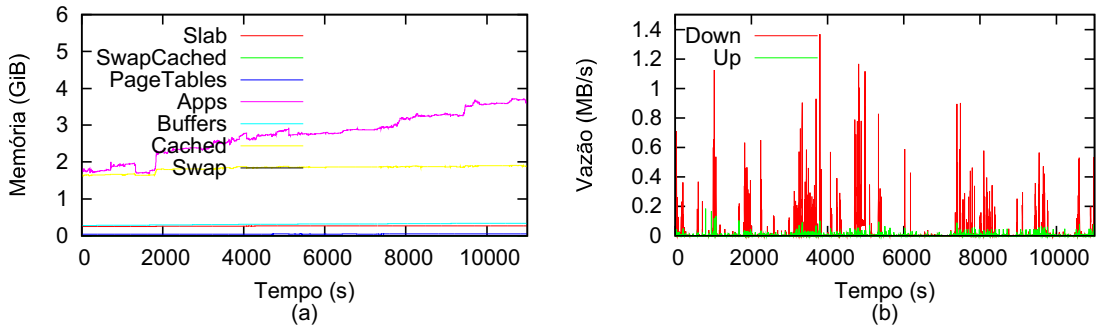

Figura 7: Memória e rede

Como exemplo da correlação de dados, uma anomalia que acontece e nitidamente é sentida por vários dos dispositivos ocorre por volta de 2000 s. Nesse instante de tempo vemos um pico no uso de CPU (Figura 6 (a)), que é perceptível com o aumento considerável do uso de memória como mostra Figura 7 (a), e de disco rígido, que tem um pico na quantidade de operações E/S e na vazão, representados respectivamente pela Figura 8 (a) e (b).

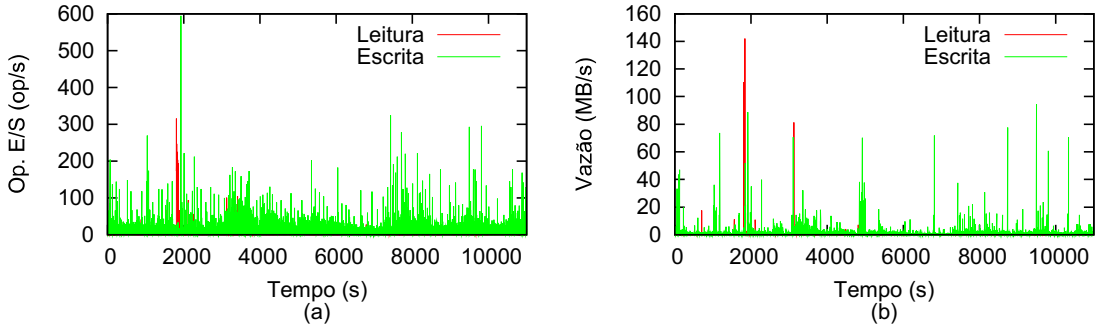

Figura 8: Vazão de E/S

3.2 Baixo Consumo de Recursos

Quanto ao consumo de recursos, independente da utilidade do sistema de monitoramento, ele nunca deve impactar no consumo de recursos do dispositivo que está sendo monitorado. É possível minimizar o impacto do monitoramento aumentando-se o tempo de captura dos dados. Porém, neste trabalho é possível capturar os dados em um intervalo pequeno e ao mesmo tempo com um consumo baixo dos recursos.

Para avaliar o baixo consumo de recursos do MSPlus, construímos um experimento de longa duração que o compara a ferramenta Munin. Nesse experimento tanto o MSPlus quanto o Munin estão com suas configurações padrão, a exceção do tempo de amostragem, que no caso do Munin foi configurado para 1 s. Os dois sistemas de monitoramento foram executados 10 vezes, por aproximadamente 6 h e 5 m (22.000 s) na máquina de testes. A máquina de testes ficou completamente ociosa durante todo o tempo dos experimentos.

O resultado com a média das 10 execuções desse comparativo entre as duas ferramentas estão representados nos gráficos das Figuras 9, 10, 11 e 12, que mostram respectivamente as métricas de CPU, memória, dispositivo de armazenamento e interface de rede. Observando essas figuras em comparação às figuras da Seção 3.1 é possível observar que a máquina encontrava-se realmente ociosa. Porém, podemos observar que ocorrem, as vezes, alguns soluços, algo que é normal devido as diversas funções que o sistema operacional desempenha mesmo quando ocioso. Um exemplo dessas atividades é a busca automática por atualizações.

Como é possível ver nos gráficos da Figura 9, MSPlus tem intrusividade bem menor que o Munin, usando aproximadamente 50 vezes menos CPU do sistema (*system* + *user*) do que Munin. Na Tabela 2 apresentamos a média do uso de CPU e o desvio padrão de ambas ferramentas. Porém vale ressaltar que esses dados mostram a soma de uso de todos os núcleos, como esta máquina tem 4 núcleos, os dados apresentados são 0,99% e 50,27% de 400% possíveis. Usando um teste-t independente o resultado produziu um valor t estatisticamente significativo para essa diferença ($t_{(10)} = 605,20, p < 0.0001$). A vantagem do MSPlus nesse caso se deve ao fato que ele não implementa *plugins* como

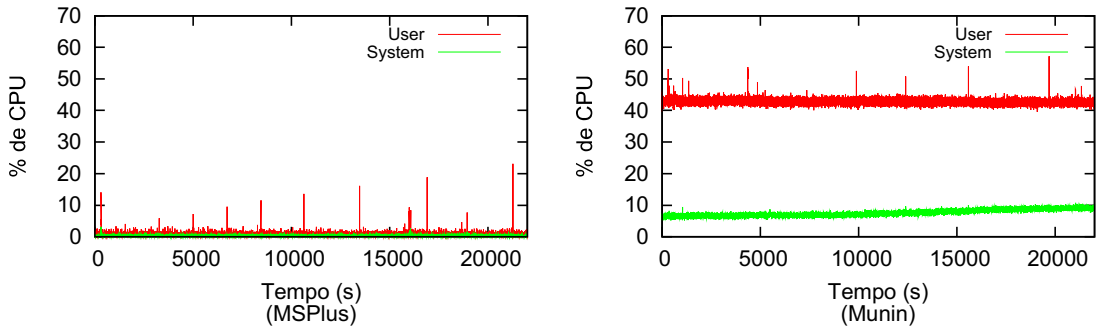

Figura 9: CPU

processos separados e não faz uso de *shell scripts* para acessar as métricas disponibilizadas pelo núcleo do Linux. Disparar um subprocesso e um interpretador de comandos tem um alto custo em termos de ciclos de CPU.

	Média	Desvio Padrão
MSPlus	0,99%	0,21
Munin	50,38%	0,15

Tabela 2: Consumo de CPU

Para avaliar o uso de memória do MSPlus e do Munin é necessário fazer uma análise de forma indireta usando os dados globais de consumo de memória obtidos. No início da execução do MSPlus o sistema utilizava em média aproximadamente 442 MB de memória com aplicativos, enquanto que no fim da execução esse valor ficou em aproximadamente 791 MB. Logo, o MSPlus usou em média 349 MB durante as 6 h e 23 m da execução. O Munin começou sua execução com média de 447 MB de memória usados por aplicativos e terminou com 582 MB utilizados. O uso total do Munin foi de 135 MB durante toda a execução. Neste critério o MSPlus usou mais recursos do sistema que o Munin. Essa situação já era esperada, pois o MSPlus usa memória como recurso para economizar disco e rede, como descrito na Seção 2.1.2. Na verdade, o consumo do Munin foi de certa forma surpreendente pois o esperado é que ele não guardasse nenhuma informação na memória principal, o que indica que o consumo observado se deve a algum vazamento de memória ou outro erro de programação.

Na Tabela 3 apresentamos o consumo médio de memória com aplicação e o desvio padrão em ambas as ferramentas.

	Média	Desvio Padrão
MSPlus	348,95 MB	10,91
Munin	135,83 MB	16,45

Tabela 3: Uso de memória por aplicações

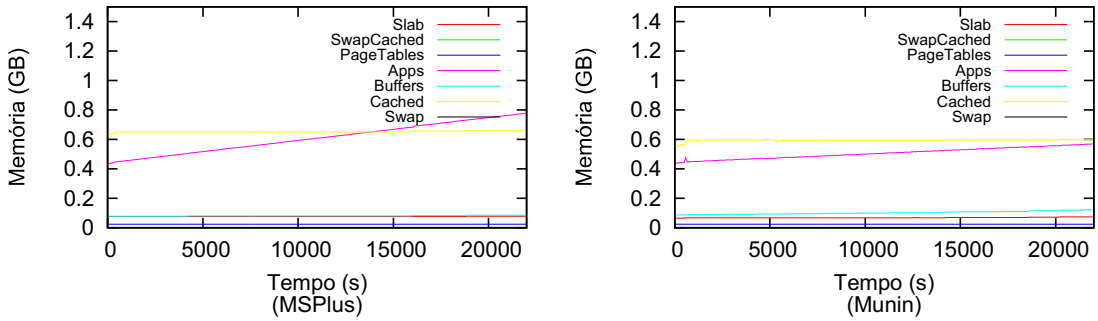

Figura 10: Memória

Na Figura 11 apresentamos o desempenho do MSPlus e do Munin em termos da vazão de E/S de disco. Percebe-se que a vazão de disco referente a leitura quase não aparece no gráfico, tendo alguns picos de desempenho no gráfico do Munin. No entanto, na vazão de escrita no disco o consumo do Munin diante do MSPlus é consideravelmente maior.

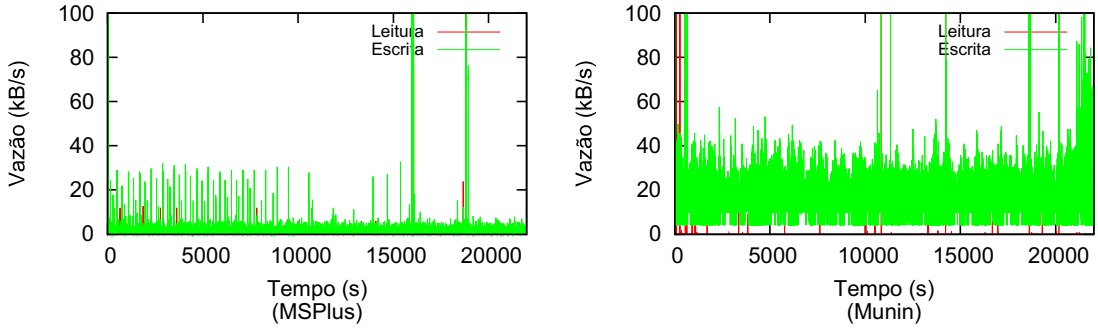

Figura 11: Vazão de disco

Na Tabela 4 apresentamos os valores médios e o desvio padrão da vazão de E/S no disco para ambas ferramentas. A diferença de desempenho entre as ferramentas é de 1,19 kB/s de leitura e 16 kB/s de escrita. Este resultado deve-se ao fato que na execução o Munin usa a memória persistente para armazenar os dados parciais utilizados para computar as métricas durante o monitoramento. Já o MSPlus utiliza apenas a memória volátil para tal função . Para a leitura, um teste-t independente não indicou uma diferença estatisticamente significativa com $p < 0,001$ ($t_{(10)} = 1,63, p = 0,15$), devido ao alto desvio padrão. Para a escrita, a diferença observada é estatisticamente significativa ($t_{(10)} = 23,56, p < 0,0001$).

A Figura 12 mostra os dados de vazão de rede apurados quando utilizamos a conexão sem fio da máquina teste. Ambas as ferramentas não utilizam conexão de rede, pois estão configuradas para apenas coletar dados de uma única máquina. Sendo assim, podemos atribuir que o pequeno consumo apresentado é referente ao uso da própria interface de rede que verifica continuamente a conexão e do sistema operacional. Neste cenário

	Leitura		Escrita	
	Média	DP	Média	DP
MSPlus	0,23 kB/s	0,68	0,85 kB/s	0,16
Munin	1,42 kB/s	2,20	17,73 kB/s	2,26

Tabela 4: Vazão de disco

simples as aplicações trabalham igualmente, pois não utilizam rede. Porém, vale salientar que se as duas ferramentas forem monitorar um aglomerado ou grade computacional, o MSPlus certamente terá consumo igual ou menor do que Munin. Isto ocorre porque o MSPlus monitora cada nó de forma independente, não tendo troca de dados em tempo de execução entre os nós monitorados. Porém, no caso do Munin, se configurado para sua função original, os nós enviam dados de monitoramento em tempo de execução pela interface de rede, o que ocasionaria um consumo maior.

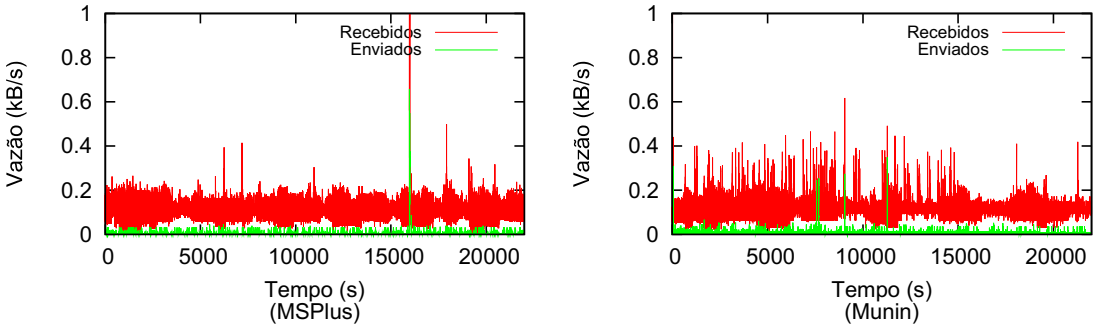

Figura 12: Vazão de rede

A diferença real é possível ser vista em maiores detalhes através da Tabela 5. Um teste-t independente indica que a diferença observada entre as duas ferramentas não e significativa com $p < 0,001$, tanto para os dados enviados ($t_{(10)} = 1,37, p = 0,20$) quanto para os recebidos ($t_{(10)} = 1,18, p = 0,28$).

	Enviados		Recebidos	
	Média	DP	Média	DP
MSPlus	0,98 kB/s	0,03	108,92 kB/s	3,17
Munin	3,42 kB/s	5,65	113,26 kB/s	11,20

Tabela 5: Vazão de rede

Até aqui apresentamos um gráfico de cada dispositivo, sendo perceptível qual é o comportamento de MSPlus em relação ao Munin. No entanto, para complementar as informações sobre o dispositivo de armazenamento de dados, apresentamos abaixo nas Figuras 13, 14 e 15, e nas Tabelas 6, 7 e 8, os gráficos e dados do tamanho médio das operações de E/S, da quantidade média das operações de E/S e da utilização do disco. Estes, junto ao gráfico de latência, representam os demais gráficos que podem ser gerados no MSPlus para o dispositivo de armazenamento. Não apresentaremos o gráfico de

latência neste caso, pois os valores observados são muito baixos, não trazendo informações relevantes.

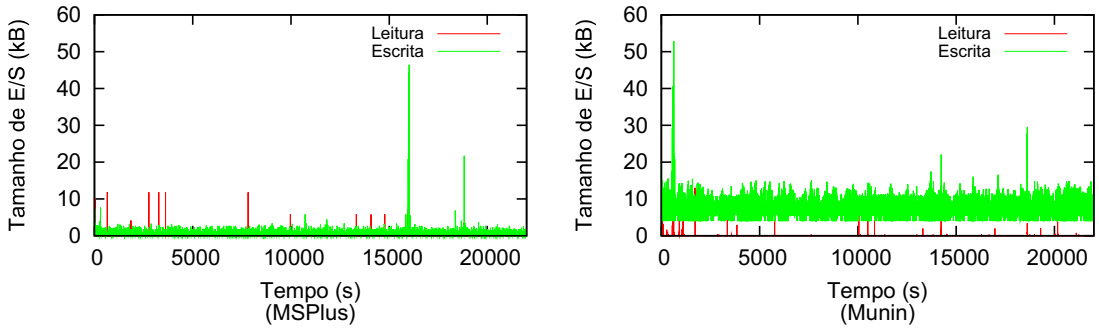

Figura 13: Tamanho médio das operações de E/S

A Figura 13 e a Tabela 6 mostram que o tamanho médio das operações do MSPlus são bem pequenos se comparado aos do Munin, Um teste-t independente indica que a diferença observada para tamanho de operação de E/S não é significativa com $p < 0,001$ para a leitura ($t_{(10)} = 0,98, p = 0,35$), mas é significativa para a escrita ($t_{(10)} = 147,32, p < 0,0001$).

	Leitura		Escrita	
	Média	DP	Média	DP
MSPlus	0,009 kB	0,007	0,30 kB	0,02
Munin	0,04 kB	0,10	7,35 kB	0,15

Tabela 6: Tamanho médio das operações de E/S

A Figura 14 e a Tabela 7 mostram a quantidade de operações de E/S feitas no intervalo de monitoramento de 1 s. Para as operações de leitura o MSPlus mostra pequena diferença em relação ao Munin. Porém, para as operações de escrita essa diferença é considerável. O mesmo é apresentado pelo teste-t independente que indica uma diferença estatisticamente significativa ($t_{(10)} = 133,18, p < 0,0001$). Porém, para a leitura não é possível observar diferenças significativas com $p < 0,001$ devido ao alto desvio padrão ($t_{(10)} = 1.55, p = 0,14$).

	Leitura		Escrita	
	Média	DP	Média	DP
MSPlus	0,04 op/s	0,12	0,12 op/s	0,009
Munin	0,15 op/s	0,19	1,52 op/s	0,032

Tabela 7: Quantidade média de operações de E/S

A Figura 15 e a Tabela 8 mostram os gráficos e dados da utilização de disco durante a execução das ferramentas. A utilização é a porcentagem de tempo que o disco ficou ocupado com leituras e escritas. Comparando as duas ferramentas, a porcentagem

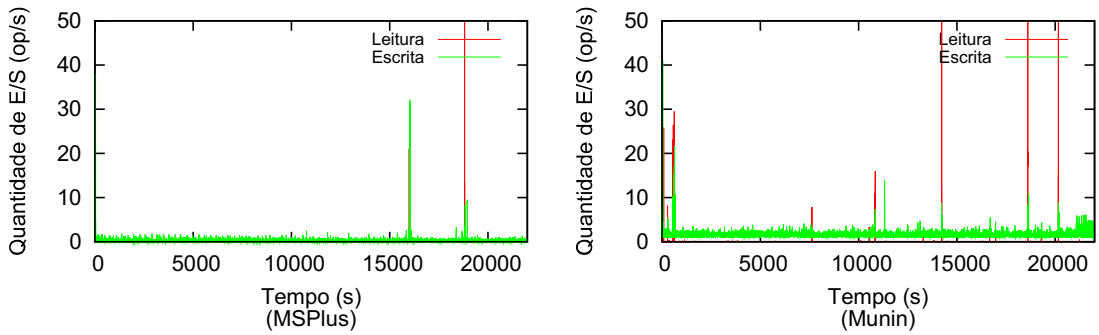

Figura 14: Quantidade média de operações de E/S

de tempo de ocupação do MSPlus são muito pequenas frente ao Munin. O teste-t independente também indica que a vantagem do MSPlus é estatisticamente significativa $(t_{(10)} = 12,42, p < 0,0001)$

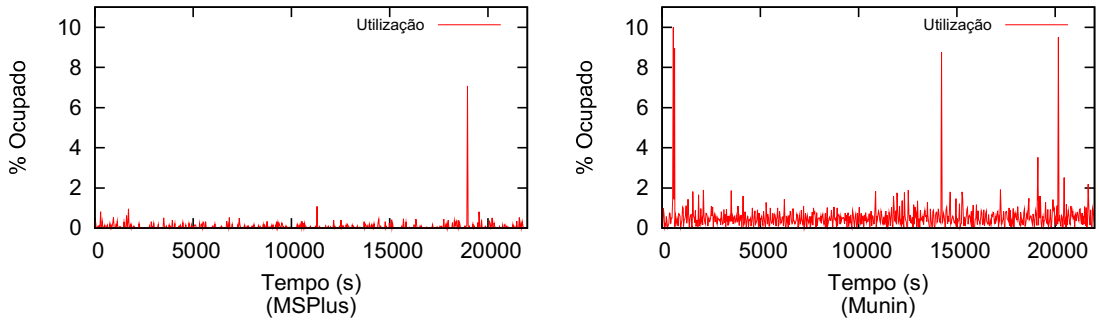

Figura 15: Utilização

	Média	Desvio Padrão
MSPlus	0,067%	0,01
Munin	0,54%	0,12

Tabela 8: Utilização

Resumindo o que apresentamos durante esta seção, o MSPlus apresentou um consumo consideravelmente inferior ao Munin nas métricas de uso de CPU e de vazão de E/S. Ele mostrou também um consumo parecido para interface de rede. Porém, apresentou um consumo maior de memória. Esse consumo de memória é esperado devido às decisões de projeto, mas este consumo pode ser minimizado com escritas regulares ao disco. Com esta opção as escritas não precisam ocorrer apenas no final da execução, podendo ser controladas por um parâmetro de linha de comando. Dessa forma, o projetista do experimento tem controle sobre como a ferramenta utiliza os recursos disponíveis. Para isso basta usar o parâmetro –*save* do MSPlus e definir o intervalo em segundos que deseja salvar os dados.

3.3 Confiabilidade dos dados

Como já mostramos até aqui, o MSPlus tem uma granularidade fina com baixo consumo de recursos. No entanto, para utilizá-lo é necessário que os dados gerados sejam confiáveis. Para isto, implementamos os *plugins* no MSPlus baseando nos do Munin que é um projeto de software livre amplamente usado. Sendo assim, seus dados são confiáveis pelo teste frequente de larga escala feito pelos seus usuários.

Para demostrar essa igualdade, realizamos um experimento com 10 execuções de aproximadamente 6 horas e 5 minutos (22000s) executando ambas ferramentas ao mesmo tempo na máquina de testes, deixando-a também ociosa durante todo o tempo dos experimentos. Levando-se em consideração que os dispositivos e métricas de monitoramento são os mesmos em ambas ferramentas, os dados devem ser iguais ou serem muito próximos. Quando dizemos muito próximos, é porque pode ocorrer variação do momento exato de captura dos dados de monitoramento dos dispositivos. Embora as ferramentas iniciem ao mesmo tempo, o modo de execução de cada uma muda, podendo ter uma diferença do tempo de captura de milésimos de segundo, tempo suficiente para que os dados já tenham mudado. Entre as causas dessa diferença de tempo estão a linguagem de programação, arquitetura e até mesmo o método utilizado na implementação de cada uma das ferramentas.

Apresentamos a seguir os dados do experimento que representam a média das 10 execuções utilizando ambas ferramentas de forma concorrente. Assim como na seção anterior, iniciaremos apresentando as figuras e tabelas com dados mais impactantes e só depois o restante. As Figuras 16, 17 e 18, representam os dados de ambas as ferramentas respectivamente para os dispositivos CPU, memória e a interface de rede. Os dados sobre o dispositivo de armazenamento serão apresentados em seguida.

A Figura 16 e a Tabela 9 mostram a soma das métricas de *user* e *system* que são as métricas que representam a utilização real de CPU. Sendo assim, percebemos uma pequena diferença que é menor que 0,005% do uso de CPU. Um teste-t independente indica que essa diferença não é estatisticamente significativa ($t_{(10)} = 0.004, p = 0,997$), sendo as duas leituras equivalentes.

	Média	Desvio Padrão
Munin	50,3831%	2,55
MSPlus	50,3875%	2,56

Tabela 9: Consumo médio de usuário e sistema em CPU

A Figura 17 e a Tabela 10 mostram que o uso de memória são semelhantes em ambas as ferramentas. Na tabela 10 consideramos a métrica que mais varia, que é a *apps*. Para esta métrica podemos observar uma pequena diferença de 2 MB no uso dos

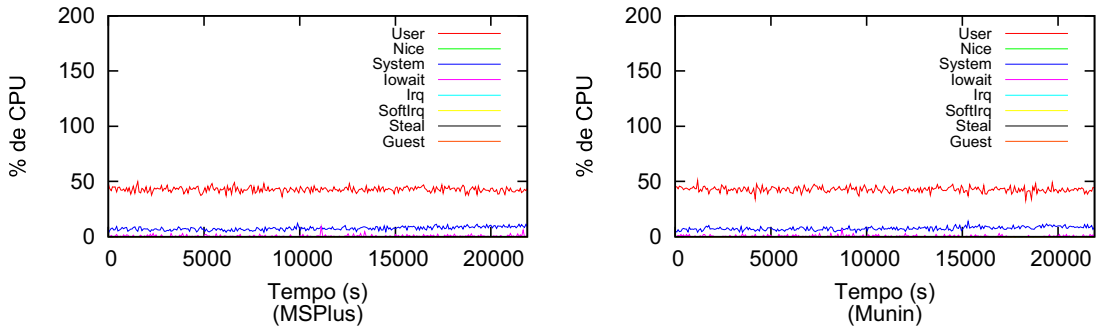

Figura 16: CPU

recursos, que é aproximadamente 0,003% do valor da métrica em questão, uma diferença não significativa de acordo com um teste-t independente ($t_{(10)} = 0,034, p = 0,97$).

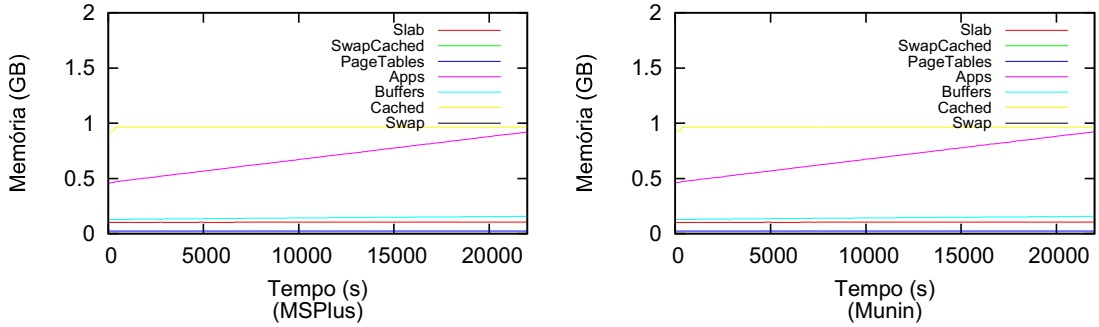

Figura 17: Memória

	Média	Desvio Padrão
Munin	710,89 MB	135,56
MSPlus	708,83 MB	135,58

Tabela 10: Consumo médio de memória pelas aplicações

A Figura 18 mostra os dados da interface de rede, que nesse experimento é uma interface sem fio para mostrar o seu comportamento mesmo durante o tempo ocioso. Quando olhamos para os gráficos de ambas ferramentas, vemos algumas diferenças visuais pequenas. A Tabela 10 mostra que essas pequenas diferenças visuais não refletem em diferença alguma nas médias obtidas. Isso ocorre porque a interface de rede é um dispositivo mais estável do ponto de vista de monitoramento. A interface de rede fornece os dados na forma de um histórico de cada byte que foi transmitido ou recebido. Sendo assim, uma diferença causada por coleta de dados para menos será compensada na próxima coleta de dados. Considerando este cenário, o total da soma de todos os dados coletados será o mesmo para ambas as aplicações, mesmo que ocorra variação durante o experimento.

Os dados relativos ao dispositivo de armazenamento estão separados. Para este experimento iremos deixar de lado os dados das métricas de latência, pois as mesmas não

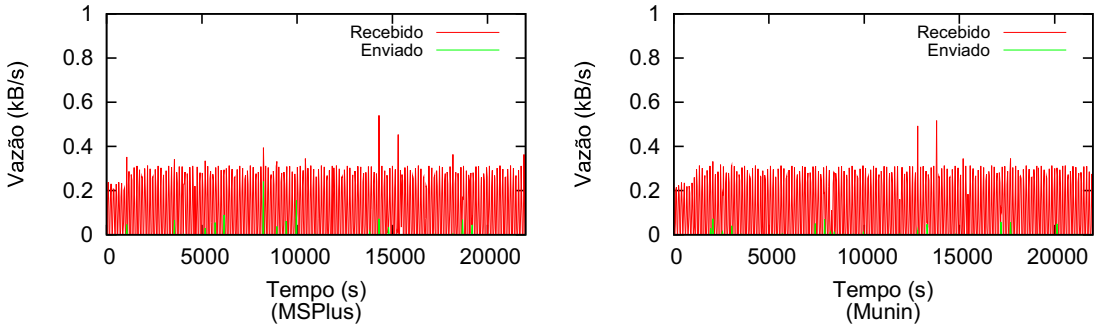

Figura 18: Vazão da interface de rede

	Enviados		Recebidos	
	Média	DP	Média	DP
MSPlus	1,010228 kB/s	12,466460	106,246148 kB/s	147,301899
Munin	1,010228 kB/s	12,466460	106,246148 kB/s	147,320842

Tabela 11: Vazão da interface de rede

mostraram atividades em ambas as ferramentas nos experimentos realizados. As demais métricas estão representadas nas Figuras 19, 20, 21 e 22, e nas Tabelas 12, 13, 14 e 15. Estes são respectivamente: o tamanho médio das operações de E/S do disco, quantidade média das operações de E/S, média da vazão de disco e média da utilização de disco.

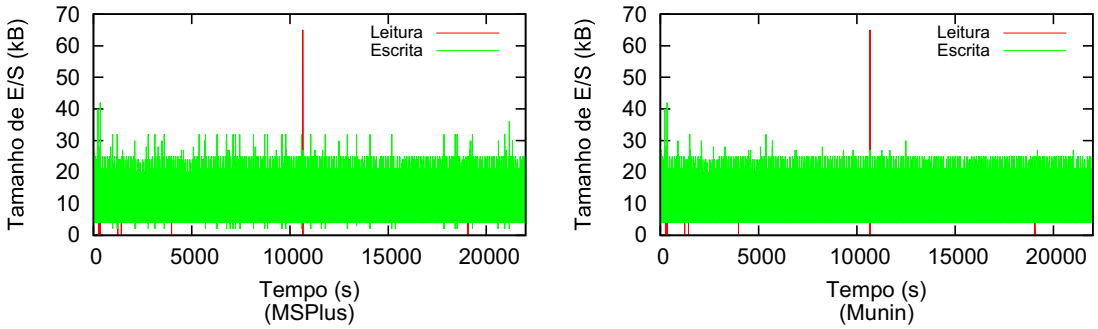

Figura 19: Tamanho médio das operações E/S

Em relação ao tamanho médio das operações de E/S do disco da Figura 19, é possível perceber algumas diferenças pequenas, mas que se mostram ainda menores olhando a Tabela 12. Para este experimento os dados de leitura estão em média iguais para as duas ferramentas, e na escrita tem uma diferença menor que 0,03 kB. Essa diferença não é estatisticamente significativa de acordo com um teste-t independente ($t_{(10)} = 0.013, p = 0,99$).

A Figura 20 e Tabela 13 apresentam a quantidade média de operações de E/S no disco. A diferença é ainda menor do que aquela observada na métrica de tamanho médio de operações de E/S, sendo igual para leitura e menor que 0,00005 requisições de escrita, sendo para indiscernível na prática.

	Leitura		Escrita	
	Média	DP	Média	DP
MSPlus	0,004682 kB	0,448275	7,401427 kB	6,910057
Munin	0,004682 kB	0,448275	7,372699 kB	6,841474

Tabela 12: Tamanho médio das operações de E/S

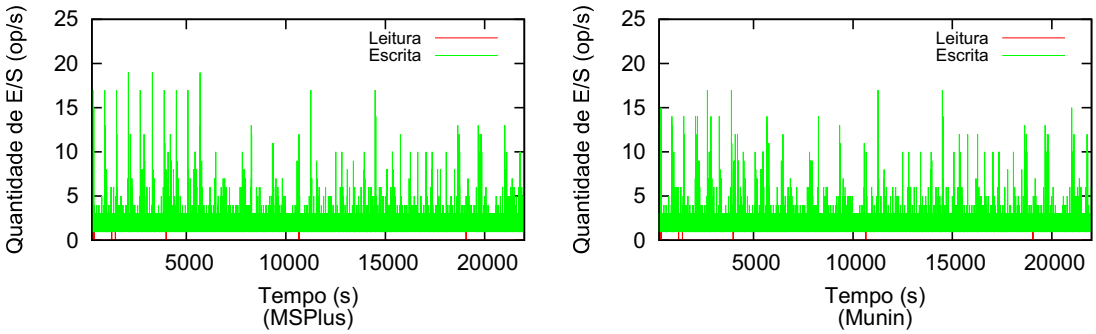

Figura 20: Quantidade média de operações de E/S

	Leitura		Escrita	
	Média	DP	Média	DP
MSPlus	0,000545 op/s	0,028600	1,518842 op/s	2,631538
Munin	0,000545 op/s	0,028600	1,518796 op/s	2,633230

Tabela 13: Quantidade média de operações de E/S

A Figura 21 e a Tabela 14 apresentam os dados de vazão de disco. Assim como nas métricas anteriores a diferença é quase que imperceptível. A média é idêntica para a leitura no experimento de ambas ferramentas e uma diferença de 5 kB na escrita, uma diferença não significativa de acordo com um teste-t independente ($t_{(10)} < 0.0001, p = 0,9999$).

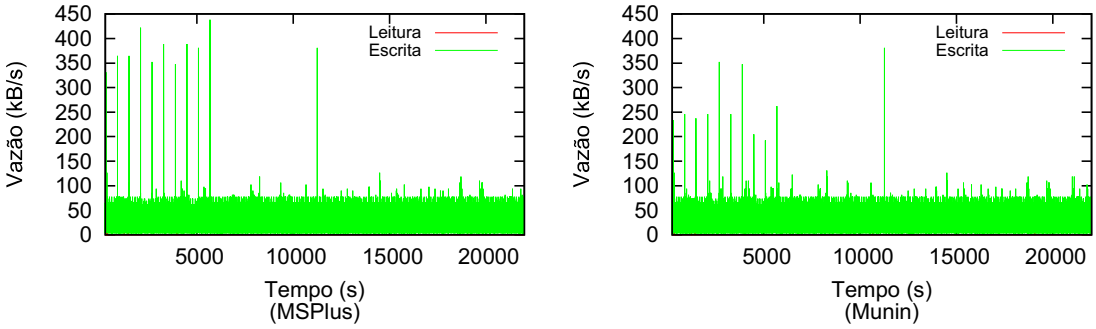

Figura 21: Vazão do disco

A Figura 22 e a Tabela 15 mostram a porcentagem de tempo de utilização do disco considerando cada segundo. Sendo assim o disco foi utilizado segundo o MSPlus por aproximadamente 4,7 milissegundos, e a diferença entre ambas as ferramentas é menor que 0,0004%, ou seja, menor que 0,004 milissegundos, uma diferença não significativa de

	Leitura		Escrita	
	Média	DP	Média	DP
MSPlus	5,399518 (kB/s)	462,905099	17139,746352 (kB/s)	28707,144494
Munin	5,399518 (kB/s)	462,905099	17136,953498 (kB/s)	28389,697581

Tabela 14: Média de vazão de disco

acordo com um teste-t independente ($t_{(10)} = 0.001, p = 0,999$).

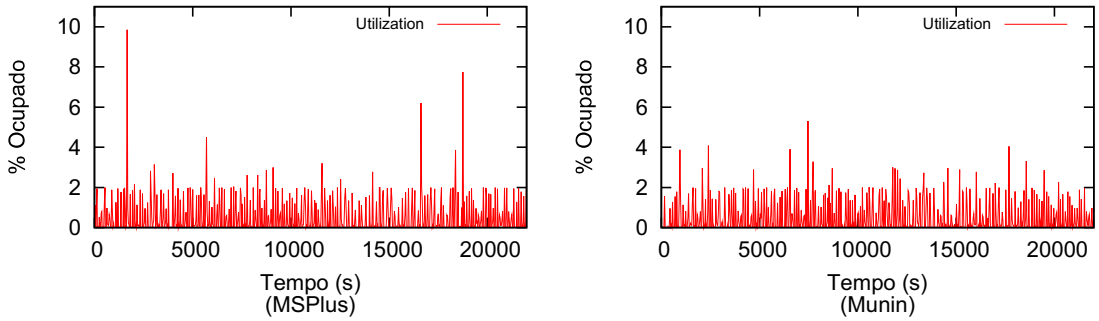

Figura 22: Utilização do disco

	Média	Desvio Padrão
Munin	0,471476%	1,081711
MSPlus	0,471112%	1,096602

Tabela 15: Média da utilização do disco

O MSPlus atende as expectativas com o monitoramento, tendo resultados bem próximos aos obtidos pelo Munin. Dessa forma podemos concluir que os resultados obtidos pelo MSPlus são tão confiáveis quanto aqueles obtidos com o Munin.

4 Caracterização

Neste capítulo apresentamos os resultados de experimentos realizados para caracterizar o funcionamento do Treplica (VIEIRA; BUZATO, 2008; VIEIRA; BUZATO, 2010) através do monitoramento feito pelo MSPlus. Para isto, em cada um dos experimentos correlacionamos os dados comportamentais de Treplica executando diversas cargas de trabalho com os dados dos principais dispositivos de cada nó do aglomerado utilizado durante os experimentos.

O Treplica é uma biblioteca com princípio de funcionamento de replicação ativa, projetada para prover uma forma simples e orientada a objetos de se construir aplicações, sendo uma solução do problema de replicação de dados altamente confiável. Treplica torna transparente a complexidade de lidar com replicação e persistência de dados, escondendo uma implementação eficiente do algoritmo Paxos (LAMPORT, 1998; LAMPORT, 2006), atrás de uma interface de programação simples de entender. Treplica funciona em aglomerados computacionais compostos por máquinas comuns sem suporte especial de hardware para tolerância a falhas.

O algoritmo de Paxos, implementado por Treplica, é uma solução completa para replicação ativa usando consenso no modelo falha-e-recuperação (LAMPORT, 1998). Nele os processos no sistema são agentes reativos que podem assumir vários papéis: proponente, receptador ou aprendiz. Estes agentes são os responsáveis por resolver o consenso e para isto executam várias rodadas, onde cada rodada possui um coordenador. Os proponentes enviam a sua proposta para o coordenador que busca o consenso baseado na regra local de um quórum de receptores e exige pelo menos $\lfloor N/2 \rfloor + 1$ receptores façam parte da rodada, onde N é o número total. O coordenador seleciona uma proposta e a envia para o quórum de receptores e estes participam votando na proposta. Os receptores votam enviando o número de rodada e a proposta aos aprendizes (LAMPORT, 2006).

Todos os experimentos foram realizados no aglomerado Maritaca da Universidade Federal de São Carlos-UFSCar, no campus de Sorocaba. Foram usados os nós Node001, Node002, Node005, Node007, Node009 e Node012, cada um deles possui um processador Intel(R) Xeon(R) CPU E5-2665 @ 2.40GHz, memória de 8GB e um HDD Sata de 1TB. Nestes nós rodavam o SO CentOS release 6.7, Python 2.6.6 e RRDtool 1.3.8.

Utilizamos uma aplicação simples com uma tabela de espalhamento que mapeia uma chave *string* para um valor. A aplicação executa de forma independente em cada réplica, com o Treplica responsável por gerenciar a comunicação entre elas. Sendo assim, quando citamos uma carga de operações, identificamos ela referindo a soma das execuções realizadas por todos os nós. Supondo que a carga global é de 75 operações por segundo,

utilizando 3 nós, cada nó gera 1/3 desse valor, ou seja 25 op/s.

Na execução dos experimentos executamos a aplicação com Treplica monitorando os nós com o MSPlus. Na Seção 4.1 apresentamos os resultados dos experimentos de uma execução com 4 cargas na qual identificamos um nó com falha. Neste experimento percebemos que o comportamento do dispositivo de armazenamento é instável, tendo comportamento variado diante de cargas diferentes. Sendo assim executamos um experimento mais profundo executando 10 vezes 11 cargas de operações em outros nós e os resultados do experimento são apresentados na Seção 4.2.

4.1 Análise de uma Falha Inesperada

Neste cenário apresentamos um experimento de aumento de carga (*speedup*). Foi feita neste experimento uma execução usando os nós Node005, Node007 e Node012 com as cargas de trabalho 75 op/s, 150 op/s, 300 op/s e 750 op/s e utilizando o sistema de arquivos ext3.

Um dos nós mudou o seu comportamento dependendo da carga de trabalho executada. Isto pode ser visto na Tabela 16, que mostra os resultados das execuções em função da carga gerada. Podemos perceber que quando o Treplica é executado com carga de trabalho de 75 op/s e 150 op/s, o número médio de operações realizadas é próximo do esperado. No entanto, com cargas maiores, como é o caso das cargas de 300 op/s e 750 op/s, em média apenas 2/3 das operações esperadas são realizadas.

Esperadas	Realizadas	Desvio Padrão
75 op/s	74,87 op/s	3,11
150 op/s	149,75 op/s	6,63
300 op/s	199,71 op/s	8,75
750 op/s	495,43 op/s	29,35

Tabela 16: Aumento de carga com falha

A Figura 23 detalha através de um gráfico o comportamento dos nós individuais nas execuções. Pode-se notar que todos os nós trabalham de forma semelhante para os dois primeiros experimentos. No entanto, com as cargas com 300 e 750 op/s, o Node007 se comporta de forma diferente dos demais e o seu desempenho diminui drasticamente, tendo resultados baixos para 300 op/s e perto de zero para a execução com carga de 750 op/s.

Para entender melhor o que aconteceu nesse experimento usamos os dados de monitoramento gerados pelo MSPlus. Nas Figuras 24, 25 e 26 apresentamos os gráficos e nas Tabelas 17, 18 e 19 os dados das execuções do ponto de vista do monitoramento de cada nó, correspondendo a cada uma das cargas geradas. Percebemos que no uso de

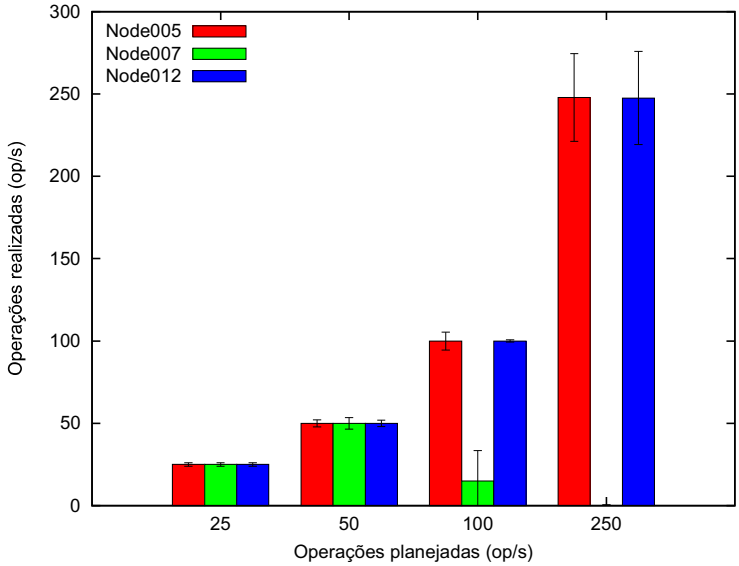

Figura 23: Aumento de carga com falha

CPU o comportamento é semelhante durante as execuções em que toda a carga gerada é atendida. Isto é, nas execuções com carga de trabalho de 75 e 150 op/s o custo de CPU foi próximo para todos os nós. Nas outras duas execuções, com 300 e 750 op/s, os nós se comportaram de formas bem distintas. O nó Node007 que teve falha e teve um desempenho menor nessas duas execuções, também consumiu menos CPU. Enquanto o nó Node005 teve um consumo relativamente maior aos demais. Esse consumo é atribuído ao nó por ter sido eleito coordenador durante as execuções de Tréplica e, com os problemas ocorrendo com o nó Node007, há um consumo maior da parte do coordenador para realizar suas tarefas e auxiliar o nó com problemas.

Carga	Node005	Node007	Node012
75 op/s	17,98 %	19,91 %	18,93 %
150 op/s	34,00 %	36,19 %	33,72 %
300 op/s	38,79 %	29,20 %	33,25 %
750 op/s	88,94 %	51,97 %	70,58 %

Tabela 17: Uso de CPU para aumento de carga com falha

A Figura 25 mostra os gráficos e a Tabela 18 os dados da vazão de rede. Pode-se perceber que os nós Node007 e Node012 se comportam de forma semelhante na execução com carga de 75 e 150 op/s, tendo menor envio e maior recebimento de dados em ambos os nós. O nó Node005 comparado aos demais tem menor vazão para recepção e maior vazão para envio de dados. Isso ocorre pois esse nó é o coordenador, embora ele receba todas as requisições dos outros nós, ele envia suas requisições além de avisos sobre qual é a decisão de consenso. Os demais nós que não são coordenadores enviam apenas as suas requisições, mas recebem as requisições de todos os outros e o aviso do coordenador. Nas execuções com 300 e 750 op/s, o comportamento do nó Node007 apresenta diferenças

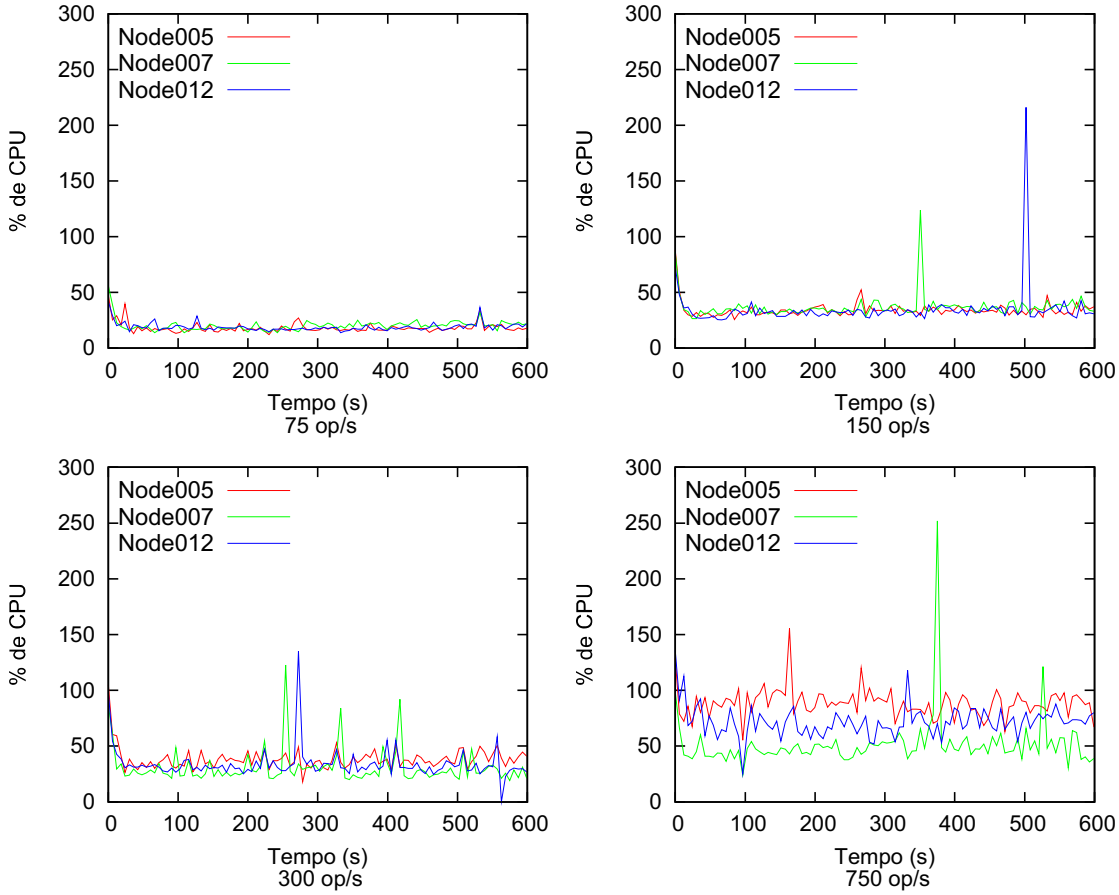

Figura 24: Uso de CPU para aumento de carga com falha

observáveis. Com 300 op/s ele busca insistentemente se recuperar, por isso a vazão de envio de dados aumenta expressivamente, enquanto o coordenador (Node005) tem uma vazão maior de recebimento. Isso também ocorre com o Node012, pois ele recebe os pedidos de recuperação do Node007, mesmo sem ser responsável por atendê-lo. Mesmo com a carga extra, os dois nós conseguem processar o volume de operações gerado, enquanto o nó Node007 processa uma pequena quantidade de operações. Com 750 op/s o nó Node007 na maioria do tempo fica travado, recebendo as mensagens normalmente, porém não enviando suas próprias propostas devido ao fato que não consegue alcançar os demais e tampouco enviando mensagens de recuperação.

Na Figura 26 e na Tabela 19 apresentamos os dados de vazão de escrita do dispositivo de armazenamento. Desconsideramos os dados da vazão de leitura porque estão muito próximos de zero, sendo impossível determinar se o uso é atribuído às execuções. Nas execuções com 75 e 150 op/s aparentemente não existem problemas, sendo consistente com os dados da vazão de rede. Com 300 op/s o nó Node007 tem uma vazão baixa comparado aos outros, isto ocorre pois ao comparar com o dispositivo de rede, o Node007 envia muitos dados, pois está enviando muitas mensagens buscando se recuperar. Mesmo com a grande quantidade de envios ele recebe um número de respostas não proporcional

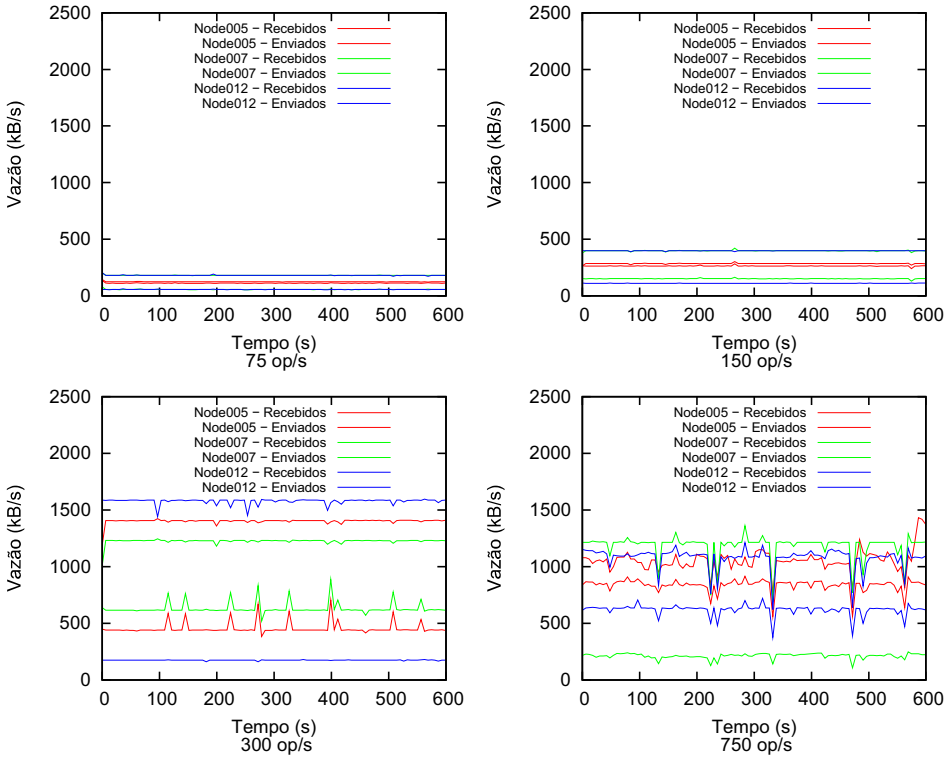

Figura 25: Vazão de rede para aumento de carga com falha

		Enviados	
Carga	Node005	Node007	Node012
75 op/s	124,67 kB/s	56,56 kB/s	56,22 kB/s
150 op/s	285,86 kB/s	151,76 kB/s	112,14 kB/s
300 op/s	453,72 kB/s	1227,72 kB/s	176,00 kB/s
750 op/s	1048,01 kB/s	211,97 kB/s	616,81 kB/s
		Recebidos	
Carga	Node005	Node007	Node012
75 op/s	113,08 kB/s	181,14 kB/s	181,17 kB/s
150 op/s	264,21 kB/s	398,13 kB/s	398,84 kB/s
300 op/s	1404,02 kB/s	629,52 kB/s	1582,72 kB/s
750 op/s	828,64 kB/s	1178,61 kB/s	1077,69 kB/s

Tabela 18: Vazão de rede para aumento de carga com falha

aos seus pedidos, possivelmente proporcional a capacidade do coordenador de responder as suas requisições. Porém, ele consegue ainda realizar uma pequena quantidade de operações a cada segundo. Já na execução com 750 op/s, o nó Node007 solicita informações em um ritmo inferior aquele observado com 300 op/s, assim ele não consegue alcançar as duas réplicas mais rápidas. Como ele recebe normalmente as decisões e mensagens que lhe são enviadas e essas decisões recebidas são gravadas em disco, ocorre uma vazão maior de escrita no disco.

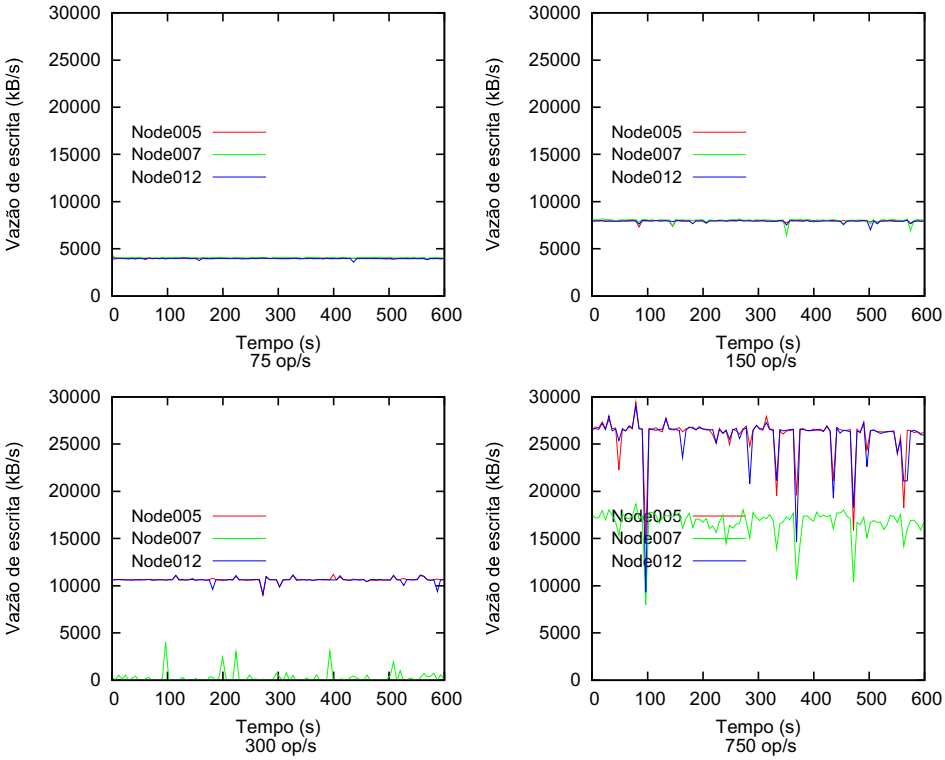

Figura 26: Vazão de escrita para aumento de carga com falha

	Escrita		
Carga	Node005	Node007	Node012
75 op/s	3973,48 kB/s	4059,37 kB/s	3969,70 kB/s
150 op/s	7940,21 kB/s	8021,13 kB/s	7925,32 kB/s
300 op/s	10612,01 kB/s	216,28 kB/s	10620,41 kB/s
750 op/s	25597,19 kB/s	16495,18 kB/s	25551,68 kB/s

Tabela 19: Vazão de escrita para aumento de carga com falha

Em resumo, neste experimento o Treplica executou normalmente com as cargas de 75 e 150 op/s em todos os nós. No entanto, perdeu desempenho considerável no nó Node007 com carga total de 300 op/s e quase não processou operações neste nó com a carga total de 750 op/s. Como o uso dos recursos depende do funcionamento de Treplica, observá-lo pode indicar a origem do problema. Porém, observando os dados de monitoramento das execuções com 300 e 750 op/s não é possível determinar responsáveis pela falha. Como a

falha ocorreu com o aumento de carga, diversos fatores podem ser o responsável. Sendo assim fizemos um novo experimento com outros nós, com maior número de cargas e variando os sistemas de arquivos.

4.2 Análise de vazão de disco

Para avaliar de forma mais abrangente o impacto dos dispositivos no desempenho de Treplica, aplicamos um novo experimento de aumento de carga (*speedup*) em que utilizamos outros nós, uma quantidade maior de cargas de trabalho e variamos os sistemas de arquivos. Mudamos os nós para garantir que em ambiente normal de execução a aplicação executaria normalmente e podemos perceber que o Treplica não sofreu com falhas. Porém, diminuiu seu desempenho com cargas de trabalho maiores e teve comportamento diferente no uso de recursos utilizando sistemas de arquivos distintos.

Foram aplicadas 11 cargas de trabalho no Treplica variando os sistemas de arquivos ext2 e ext3, usando três nós. As cargas utilizadas são: 75 op/s, 150 op/s, 300 op/s, 750 op/s, 1500 op/s, 2250 op/s, 3000 op/s, 3750 op/s, 4500 op/s, 5250 op/s e 6000 op/s. Os nós utilizados foram: os nós Node001, Node002 e Node009. Procuramos nós com mesmo desempenho para evitar que diferenças de comportamento escondam a informação que nos interessa.

Todos os dados apresentados a seguir são médias e desvios padrão de 10 execuções feitas nos três nós, com as 11 cargas para cada um dos sistemas de arquivos. Conforme citamos acima, quando referimos a cargas de trabalho, este valor é o total de operações esperadas para executar a cada segundo. Sendo assim, para este cenário com 3 nós, cada nó representa 1/3 da carga de op/s.

Na Tabela 20 apresentamos o total de operações realizadas por Treplica em cada carga de trabalho para cada sistema de arquivo. Pode-se ver que o ext3 tem uma melhor média de desempenho em 10 das 11 cargas utilizadas, sendo mais eficiente do que o ext2 até mesmo para pequenas cargas de trabalho. O ext3 foi capaz de concluir todas as operações planejadas para cargas de 75 e 150 op/s durante todo o tempo de execução, sem variação. No entanto, este sistema de arquivo apresentou em várias execuções de cargas maiores um desvio padrão alto.

Na Figura 27 apresentamos o gráfico que representa a Tabela 20. No geral é possível ver que após uma carga de 150 op/s, o Treplica começa a ter perda de desempenho, não atendendo todas as operações esperadas. A perda se mantém em no máximo 10% das operações esperadas até a carga de 1500 op/s, que não realiza em média 20% para a carga 2250 op/s. Após isto, mesmo para valores altos, a quantidade de op/s atendidas passa a se manter. Ou seja, a partir de 3000 op/s o número de operações realizadas fica constante em 2200 para ext2 e 2250 para ext3. Estes valores mostram o limite de operações que o

	ext2		ext3	
Carga	Média	Desvio Padrão	Média	Desvio Padrão
75 op/s	74,99 op/s	0,004	75,0 op/s	0,0
150 op/s	146,84 op/s	0,75	150,0 op/s	0,0
300 op/s	278,10 op/s	8,03	277,50 op/s	71,15
750 op/s	588,36 op/s	7,06	745,55 op/s	1,60
1500 op/s	1100,79 op/s	23,42	1396,08 op/s	7,02
2250 op/s	1610,63 op/s	16,75	1830,74 op/s	16,05
3000 op/s	1984,81 op/s	32,58	2115,67 op/s	254,12
3750 op/s	2187,56 op/s	23,67	2431,45 op/s	57,46
4500 op/s	2212,38 op/s	17,16	2252,28 op/s	15,69
5250 op/s	2198,26 op/s	18,02	2221,74 op/s	142,35
6000 op/s	2206,14 op/s	13,00	2375,23 op/s	9,63

Tabela 20: Aumento de carga por sistema de arquivos

Treplica tem em relação a quantidade de op/s que são possíveis de ser realizadas neste ambiente. Um fato interessante a se observar é que o valor máximo de operações realizadas é muito próximo para os dois sistemas de arquivos.

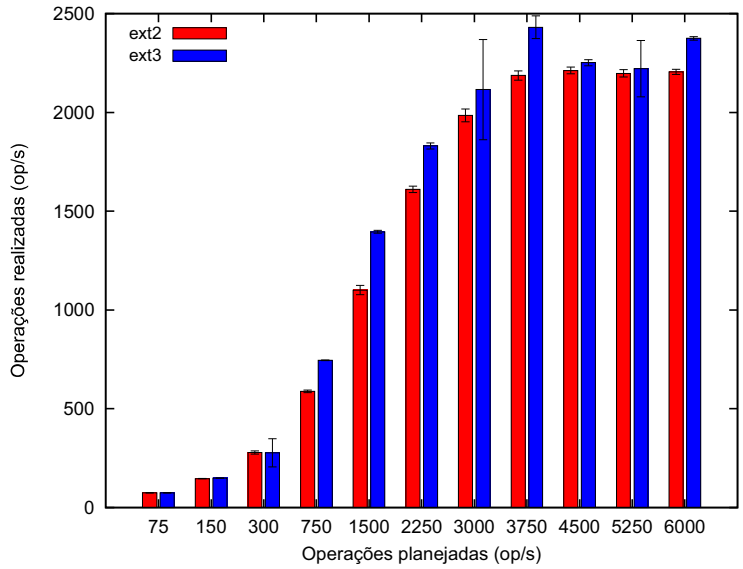

Figura 27: Aumento de carga por sistema de arquivos

Nas figuras a seguir apresentamos os dados do monitoramento coletados durante as execuções de Treplica, comparando os dois sistemas de arquivos diferentes. Iniciamos com o dispositivo de CPU representado na Figura 28 que mostra o total de utilização de usuário somado ao uso do sistema operacional. Podemos ver que embora uma execução com ext3 tenha melhor desempenho, o consumo é consideravelmente maior de CPU, se comparado ao consumo nas execuções utilizando o sistema de arquivos ext2.

Na Figura 29 apresentamos a média de uso da memória, representada pela métrica relacionada ao consumo dos aplicativos executando no ambiente. O comportamento para

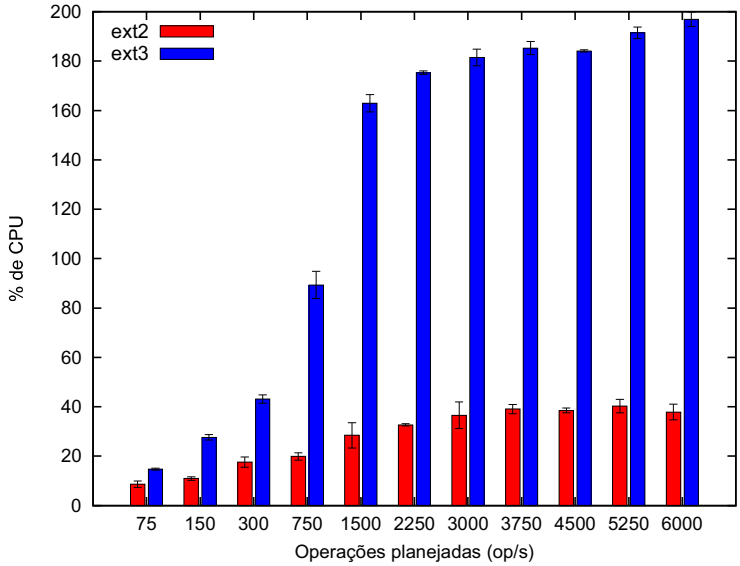

Figura 28: Utilização de CPU por sistema de arquivos

os experimentos com cargas menores usando ext3 é semelhante àquele das execuções com ext2. No entanto, a partir da carga com 750 op/s o uso do recurso mostra um expressivo consumo do ext3 em relação ao ext2, similar ao que ocorre com CPU. Quanto ao experimento de 5250 op/s com ext2, aconteceu um soluço durante a execução de um dos experimentos que afetou a média de consumo e o desvio padrão.

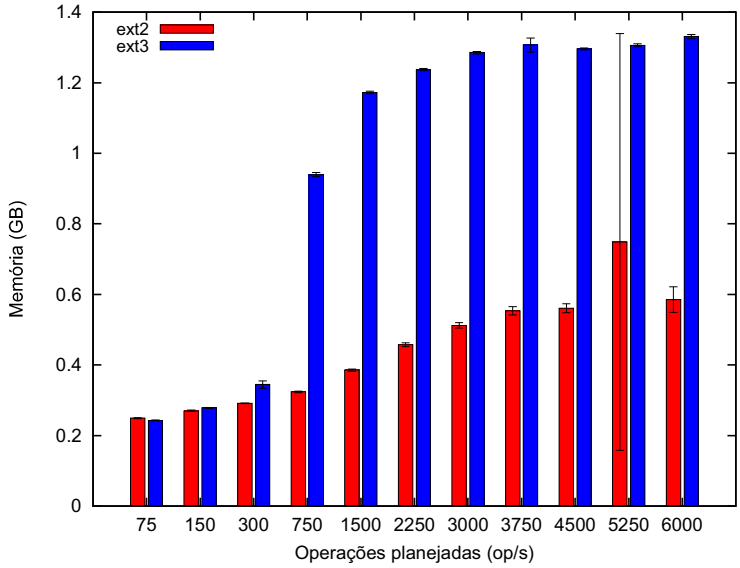

Figura 29: Média de uso de memória por sistema de arquivos

Quando ao uso de rede, a Figura 30 apresenta a média de kilobytes recebidos e enviados durante todas as execuções. Os gráficos são muito parecidos tanto para o envio quanto para o recebimento dos dados, mostrando valores bem próximo do dobro para os dados recebidos em relação ao dados enviados. Essa relação entre dados enviados e

recebidos é consistente com os dados apresentados na Figura 25 na seção anterior. Nas execuções com ext2 o envio e recebimento aumentam de forma proporcional ao tamanho da carga até 2250 op/s, após isto eles mantém-se estáveis. No entanto, para execuções com ext3 esses valores crescem proporcionalmente a carga até 5250 op/s.

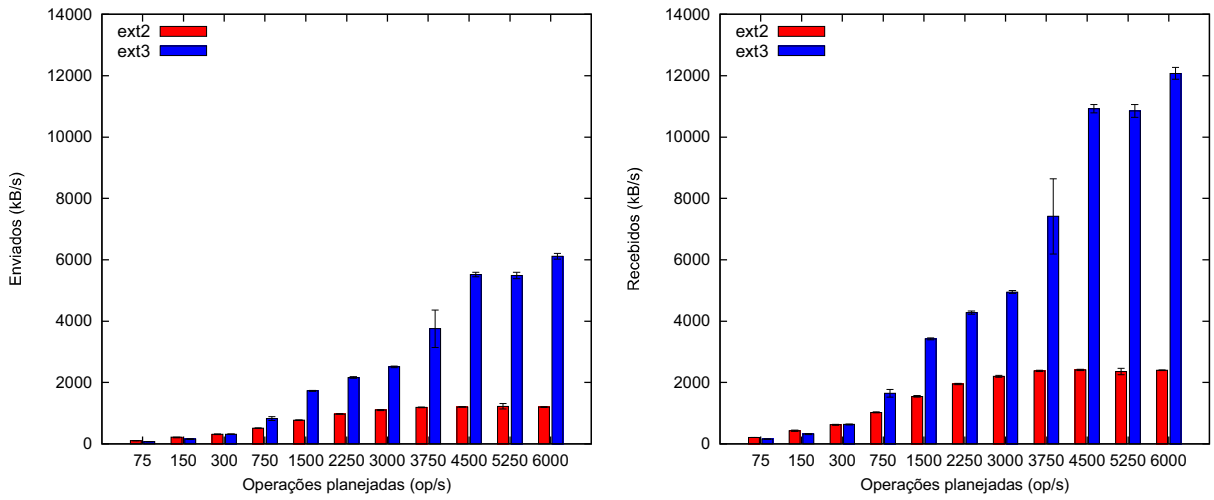

Figura 30: Vazão de rede por sistema de arquivos

As Figuras 31, 32, 33 e 34 apresentam algumas métricas de uso do disco. Cada figura representa respectivamente o tamanho médio das operações, a quantidade média de operações, a vazão e a utilização do disco. Os gráficos que representam estas métricas mostram apenas valores para escrita, pois o Treplica não faz leitura de dados durante a sua operação sem falhas. No caso da média de escrita nas execuções com ext2, em um dos experimentos ocorreu uma anomalia durante sua execução, ocasionando o grande desvio padrão observado. Vemos que existe um tamanho médio para pequenas cargas e conforme esta aumenta, o tamanho médio de escrita também aumentou, voltando a se estabilizar após a carga com 3750 op/s. No entanto, nas execuções com ext3 ocorre o contrário, temos uma média maior para pequenas cargas e com o aumento delas diminuiu o tamanho médio de escrita no disco. Após uma carga de 1500 op/s praticamente manteve-se a média de 5 kB de escrita para todos os valores de carga.

Na Figura 32 apresentamos os dados referentes à quantidade média de operações realizadas durante os experimentos. Para as execuções com ext2 a quantidade média de operações de escrita é constante, o que explica o aumento do tamanho da escrita pois é uma forma de aumentar a vazão. No caso das execuções com ext3 ocorre o aumento no número de operações de escrita, uma vez que o tamanho diminui conforme aumenta a carga.

A Figura 33 mostra a vazão do dispositivo de armazenamento. Podemos dizer que ele reflete o produto dos números dos gráficos de quantidade média de operações

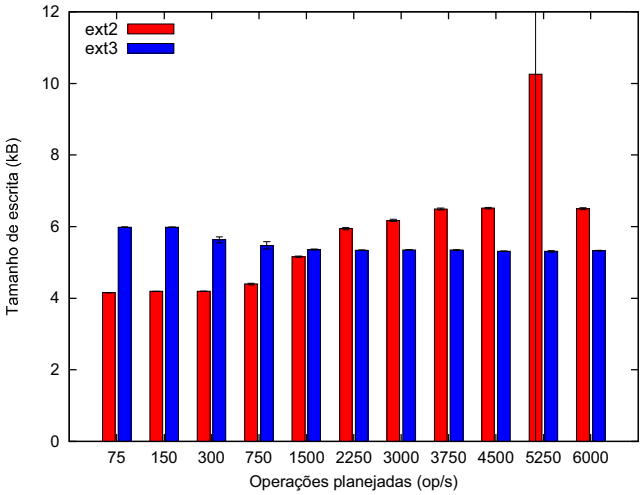

Figura 31: Tamanho médio de E/S por sistema de arquivos

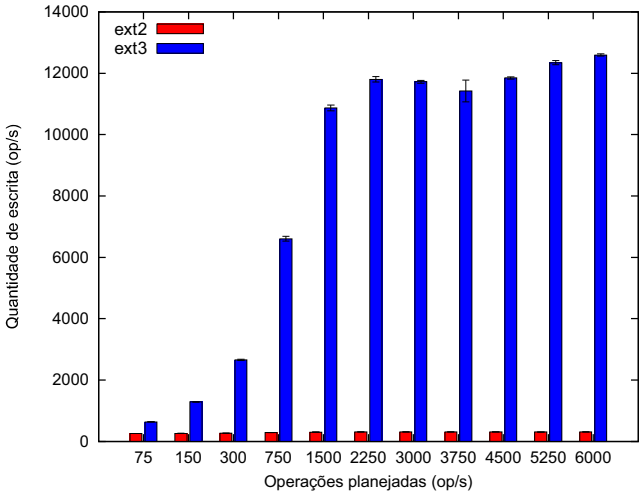

Figura 32: Quantidade de operações de E/S por sistema de arquivos

com o tamanho médio de operações. Desta forma, nas execuções com ext2 a vazão tem valores praticamente constantes. Porém, nas execuções com ext3 a diferença expressiva da quantidade de operações reflete também na vazão, que chega ser 10 vezes maior em relação às execuções com ext2.

A Figura 34 representa a utilização do dispositivo de armazenamento. Os dados mostram a porcentagem de tempo que o dispositivo ficou ocupado com as operações de E/S. Percebemos que neste caso as execuções com ext2 ficam mais de 90% do tempo ocupadas, enquanto as execuções com ext3 a ocupação não passa de 40% para qualquer um dos experimentos.

Nesta seção podemos perceber que o Treplica funcionou normalmente até mesmo com cargas maiores às utilizadas no experimento da Seção 4.1. Desta forma, podemos atribuir a falha a algum dispositivo do nó Node007. No entanto, não á possível definir

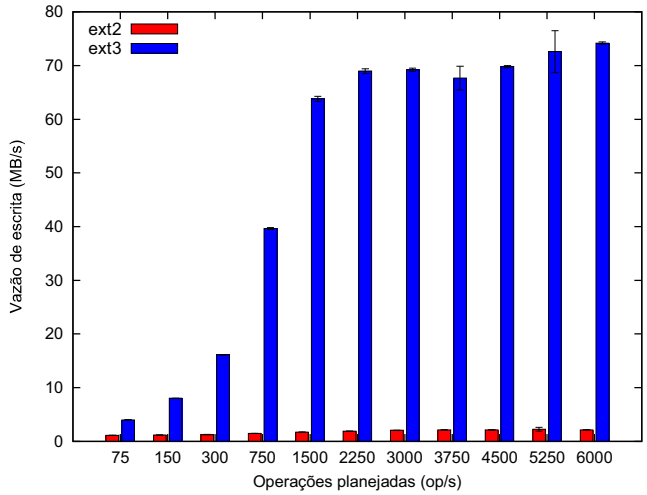

Figura 33: Vazão de E/S por sistema de arquivos

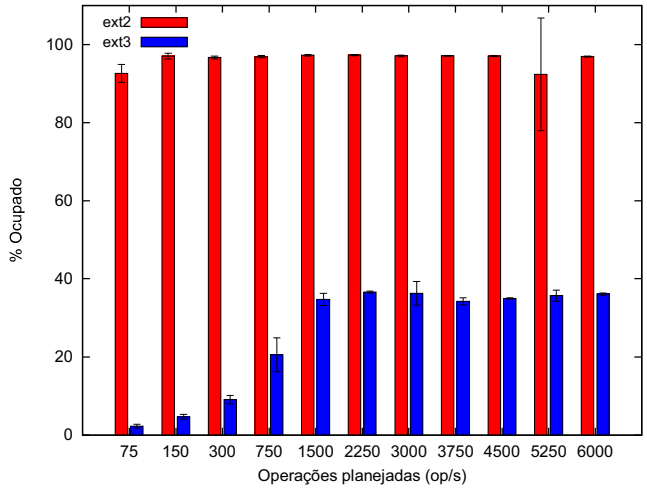

Figura 34: Utilização de disco por sistema de arquivos

qual o dispositivo responsável. Quanto ao desempenho de Treplica é possível ver outras características interessantes. Podemos perceber que ele tem melhor média de execução com ext3. Embora essa diferença seja pequena a limitação de execução em ambos tem causas diferentes. Com ext3 é possível afirmar que o disco não é o dispositivo diretamente responsável por limitar as execuções de Treplica, mas pode ser o responsável de causar o consumo maior de outros recursos, sendo assim, seria importante aumentar o tamanho de escrita pelo Treplica em disco para diminuir a quantidade de escrita. Acerca do uso de CPU pode ser um ponto de contenção, uma vez que se a aplicação não utiliza paralelismo e as tarefas são feitas por um único núcleo de CPU, pode ocorrer limitações de CPU. Quanto as execuções de Treplica com sistema de arquivos ext2 a utilização de disco está no limite praticamente com todas as cargas de trabalho. Isso mostra que o disco é um limitante com este sistema de arquivos, mas o Treplica consegue compensar isso.

Os resultados mostrados neste capítulo mostram que o MSPlus pode facilitar na

identificação de fatores impactantes do algoritmo distribuído no sistema computacional. E assim, contribuir em sanar dúvidas quanto a utilização de recursos por parte de um algoritmo distribuído.

Conclusão

Neste trabalho propusemos utilizar um sistema de monitoramento para dar suporte à execução experimental de algoritmos distribuídos, com o objetivo de relacionar o estado dos dispositivos com os dados da execução e, desta forma, aplicar uma análise do uso de recursos do aglomerado pela aplicação. Por ser um caso específico, precisávamos de uma ferramenta que fosse capaz de coletar informações com pequeno intervalo, com baixa intrusividade e realizasse o armazenamento total dos dados.

Como não encontramos nenhuma ferramenta que atendesse nossas necessidades, desenvolvemos o MSPlus. O resultado que conseguimos é uma ferramenta capaz de fazer coleta de dados com resolução de segundos por longo período e com uso mínimo dos recursos do ambiente. Essas características foram validadas através de experimentos no qual monitoramos uma máquina de um usuário comum e podemos observar como a granularidade fina pode ajudar tanto na percepção de uma anomalia como na identificação do tempo exato em que ocorreu ou iniciou o problema. Pudemos ver também o quão eficiente o MSPlus é na utilização de recursos, tendo menor consumo de recursos de CPU, interface de rede e dispositivo de armazenamento quando comparado ao consumo do Munin. Por fim, mostramos o quanto os dados do MSPlus são confiáveis tendo os dados semelhantes ao Munin em uma execução concorrente.

Com uma ferramenta de acordo com nossas necessidades, executamos uma aplicação com Treplica e monitoramos os nós com o MSPlus. Foram feitos experimentos com aumento de carga (*speedup*), em um primeiro momento um dos nós dependendo da carga de trabalho mudou o seu comportamento. Como não identificamos a causa e o Treplica poderia estar falhando, executamos outro experimento com maior abrangência em nós diferentes. Neste segundo caso, o Treplica funcionou normalmente, porém dependendo da carga utilizada com limitações. Analisando estas limitações caracterizamos o funcionamento da aplicação e identificamos alguns pontos importante para melhoria do desempenho do Treplica.

Como criamos uma aplicação que auxilia na caracterização de algoritmos distribuídos com capacidade de captar os dados em um intervalo pequeno e com baixo consumo, acreditamos que contribuímos com uma ferramenta com características um tanto diferentes de outras ferramentas de monitoramento existentes. Além disso, mesmo com os experimentos voltados para a falha, conseguimos apontar os possíveis pontos que limitam o Treplica a ter um desempenho ainda melhor.

Quanto aos trabalhos futuros, uma ferramenta como essa abre espaço para o estudo futuro de técnicas de análise preditiva de dados de monitoramento em busca de sinais de

falha e potenciais problemas de desempenho.

Publicações

MSPlus: Monitoramento de Algoritmos Distribuídos com Alta Granularidade. Na *Sessão de Ambientes de Software para Sistemas Distribuídos e Desempenho, Escalabilidade e Tolerância a Falhas* do Salão de Ferramentas. Evento organizado pelo Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC) em Maio de 2015 na cidade de Vitória.

Referências

ANDREOZZI, S. et al. GridICE: A monitoring service for grid systems. *Future Generation Computer Systems*, Elsevier, v. 21, n. 4, p. 559–571, 2005. Citado na página 28.

BAKER, M.; SMITH, G. GridRM: An extensible resource monitoring system. In: IEEE. *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*. [S.l.], 2003. p. 207–214. Citado na página 28.

BALATON, Z.; GOMBÁS, G. Resource and job monitoring in the grid. In: *Euro-Par 2003 Parallel Processing*. [S.l.]: Springer, 2003. p. 404–411. Citado na página 28.

BALATON, Z. et al. From cluster monitoring to grid monitoring based on GRM. In: SPRINGER-VERLAG. *Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*. [S.l.], 2001. p. 874–881. Citado 4 vezes nas páginas 24, 28, 29 e 31.

BALIŚ, B. et al. Monitoring grid applications with grid-enabled OMIS monitor. In: SPRINGER. *Grid Computing*. [S.l.], 2004. p. 230–239. Citado na página 28.

BONNASSIEUX, F.; HARAKALY, R.; PRIMET, P. MapCenter: an open grid status visualization tool. In: *Proceedings of the ICSA 15th International Conference on Parallel and Distributed Computing Systems*. [S.l.: s.n.], 2002. Citado na página 28.

COOKE, A. et al. R-GMA: An information integration system for grid monitoring. In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*. [S.l.]: Springer, 2003. p. 462–481. Citado na página 29.

CZAJKOWSKI, K. et al. Grid information services for distributed resource sharing. In: IEEE. *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*. [S.l.], 2001. p. 181–194. Citado na página 27.

DINDA, P. A. et al. The architecture of the Remos system. In: IEEE. *High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on*. [S.l.], 2001. p. 252–265. Citado na página 28.

EDGEWALL SOFTWARE. *Munin: the monitoring tool*. 2002. <http://munin-monitoring.org/>. Citado 2 vezes nas páginas 29 e 34.

GUNTER, D.; TIERNEY, B. NetLogger: A toolkit for distributed system performance tuning and debugging. In: IEEE. *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*. [S.l.], 2003. p. 97–100. Citado 3 vezes nas páginas 19, 28 e 32.

LAMPORT, L. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 16, n. 2, p. 133–169, 1998. Citado na página 71.

LAMPORT, L. Fast paxos. *Distributed Computing*, Springer, v. 19, n. 2, p. 79–103, 2006. Citado na página 71.

LEGRAND, I. et al. MonALISA: A distributed monitoring service architecture. In: *Computing in High Energy Physics*. [S.l.: s.n.], 2003. Citado na página 28.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, v. 30, p. 817–840, 2004. Citado 3 vezes nas páginas 19, 27 e 30.

MILLER, B. P. et al. The Paradyn parallel performance measurement tool. *Computer*, IEEE, v. 28, n. 11, p. 37–46, 1995. Citado na página 28.

OETIKER, T. *RRDtool - Round Robin Database Tool*. 2012. <http://oss.oetiker.ch/rrdtool/>. Citado 2 vezes nas páginas 35 e 52.

RIBLER, R. L. et al. Autopilot: Adaptive control of distributed applications. In: IEEE. *High Performance Distributed Computing, 1998. Proceedings. the Seventh International Symposium on*. [S.l.], 1998. p. 172–179. Citado na página 27.

SACERDOTI, F. D. et al. Wide area cluster monitoring with Ganglia. In: IEEE. *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*. [S.l.], 2003. p. 289–298. Citado 3 vezes nas páginas 19, 29 e 30.

SCHNORR, L. M. *Monitoramento de Aplicações Grid: Uma análise entre ambientes de programação e ferramentas de monitoramento*. [S.l.], 2005. Disponível em: <http://www.inf.ufrgs.br/~schnorr/download/publication/ti2005schnorr.pdf>. Citado na página 33.

SMITH, M. H. *Red Hat Enterprise Linux Deployment Guide*. 2007. <https://www.centos.org/docs/5/html/5.1/Deployment_Guide/s2-proc-meminfo.html>. Citado na página 50.

SMITH, W. et al. A grid monitoring architecture. 2001. Citado 2 vezes nas páginas 19 e 25.

SOTTILE, M. J.; MINNICH, R. G. Supermon: A high-speed cluster monitoring system. In: IEEE. *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*. [S.l.], 2002. p. 39–46. Citado 3 vezes nas páginas 19, 29 e 33.

STELLING, P. et al. A fault detection service for wide area distributed computations. *Cluster Computing*, Springer, v. 2, n. 2, p. 117–128, 1999. Citado na página 28.

TANENBAUM, A. S.; STEEN, M. V. *Distributed systems*. [S.l.]: Prentice-Hall, 2007. Citado na página 23.

TESSER, R. K. *Monitoramento On-line em Sistemas Distribuídos: Mecanismo Hierárquico Para Coleta de Dados*. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciência da Computação/UFRGS, 2011. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/37183/000819444.pdf>. Citado 5 vezes nas páginas 19, 23, 24, 25 e 29.

TIERNEY, B. et al. A grid monitoring architecture. *Recommendation GWD-I (Rev. 16*, 2002. Citado 3 vezes nas páginas 11, 25 e 26.

TIERNEY, B. et al. A monitoring sensor management system for grid environments. *Cluster Computing*, Springer, v. 4, n. 1, p. 19–28, 2001. Citado na página 28.

TIERNEY, B. et al. The NetLogger methodology for high performance distributed systems performance analysis. In: *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*. [S.l.: s.n.], 1998. p. 260–267. ISSN 1082-8907. Citado 2 vezes nas páginas 29 e 32.

TRUONG, H.; FAHRINGER, T. SCALEA-G: A unified monitoring and performance analysis system for the grid. *Scientific Programming*, IOS Press, v. 12, n. 4, p. 225–237, 2004. Citado na página 29.

VIEIRA, G. M. D.; BUZATO, L. E. Treplica: ubiquitous replication. In: *SBRC'08: Proc. of the 26th Brazilian Symposium on Computer Networks and Distributed Systems*. [S.l.: s.n.], 2008. Citado na página 71.

VIEIRA, G. M. D.; BUZATO, L. E. Implementation of an object-oriented specification for active replication using consensus. In: . [S.l.]: Technical Report IC-10-26, Institute of Computing, University of Campinas, 2010. Citado na página 71.

WOLSKI, R.; SPRING, N. T.; HAYES, J. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, Elsevier, v. 15, n. 5, p. 757–768, 1999. Citado na página 28.

ZANIKOLAS, S.; SAKELLARIOU, R. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, Elsevier, v. 21, n. 1, p. 163–188, 2005. Citado 5 vezes nas páginas 19, 24, 25, 26 e 27.