Rodolfo Santos Vera Cruz

COLETA DE DADOS PARA MINERAÇÃO DE PROCESSOS DE *MIDDLEWARE*

Trabalho de Graduação

Universidade Federal de Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~secgrad

RECIFE
2016

Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

Rodolfo Santos Vera Cruz

COLETA DE DADOS PARA MINERAÇÃO DE PROCESSOS DE *MIDDLEWARE*

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Nelson Souto Rosa*

RECIFE
2016

Agradecimentos

Primeiramente, gostaria de agradecer à Deus, por me proporcionar saúde e força para realizar meus sonhos e enfrentar as dificuldades do dia a dia. Gostaria de agradecer também à minha família, pelo enorme suporte que me deram durante toda minha vida, fazendo com que meu principal foco fosse a educação. Agradeço minha namorada pelo apoio e motivação constantes, me proporcionando momentos de descontração e tranquilidade, sempre que possível. Também agradeço à todos os meus amigos de sala, que sempre me ajudaram tornando o curso mais prazeroso. Por fim, gostaria de agradecer à todos os professores do Centro de Informática, por terem me passado todo o conhecimento da área que possuo até hoje. Dentre os professores, agradeço, especialmente ao prof. Nelson Souto Rosa, meu orientador durante o desenvolvimento deste trabalho, o qual sempre se mostrou interessado no meu aprendizado, estando disponível para sanar minhas dúvidas.

Algumas pessoas acham que foco significa dizer sim para a coisa em que você vai se focar. Mas não é nada disso. Significa dizer não às centenas de outras boas ideias que existem. Você precisa selecionar cuidadosamente.

—STEVE JOBS

Resumo

A utilização de softwares atualmente está bastante difundida. É muito fácil encontrar pessoas de diversas partes do mundo que possuem algum contato diário com serviços de software, como Google, *sites* de compra *online* ou até mesmo algum jogo para dispositivos móveis. Esta grande facilidade de acesso a esses serviços tem criado uma alta demanda para empresas do ramo.

Por conta desta crescente demanda, e com o objetivo final de prestar um bom serviço para seus clientes, muitas empresas tem adotado a utilização de sistemas distribuídos, os quais conseguem atender uma grande quantidade de usuários com qualidade.

Junto com a crescente utilização dos sistemas distribuídos, vem uma grande preocupação a respeito da manutenção desses sistemas. Fazer manutenção em aplicações distribuídas é uma tarefa complexa, pois ele consiste de vários pedaços de softwares espalhados em diferentes máquinas e que podem ter sido desenvolvidos por diferentes grupos, usando diferentes linguagens de programação. Por este motivo, a utilização de ferramentas que auxiliem a manutenção de sistemas distribuídos se faz cada vez mais útil e necessário.

Uma boa alternativa para auxiliar essa manutenção é a mineração de processos, a qual consegue extrair um modelo explícito do processo a partir de eventos registrados em um arquivo de log. Por meio deste modelo é possível verificar como os módulos de um sistema estão se relacionando na realização de determinada tarefa, facilitando, desta forma, a identificação de falhas e, consequentemente, a manutenção do sistema.

Neste trabalho será apresentada uma abordagem para a realização de uma das etapas presentes em um sistema de *trace* para *middleware*, a coleta de dados. Esta etapa tem como função principal coletar os eventos realizados pela execução do *middleware*, ordená-los e formatá-los, gerando um log de atividades do sistema para uma posterior análise em uma ferramenta de mineração de processos.

Palavras-chave: Sistemas Distribuídos, *Middleware*, Ordenação de Eventos, Log, Interpolação

Abstract

Nowadays, the softwares are in everywhere. It is very easy to see people from different parts of the world that have some daily contact with any type of service provided by softwares, like Google, shopping websites, or mobile games. The easy accessibility to this types of services has been creating a huge demand for software enterprises.

Because of this high demand, and with the goal to provide a good service for their clients, a bunch of enterprises has been utilizing the distributed systems, which can support a lot of users without lost the quality.

The development of distributed systems brings with it the concern about maintenance of this type of system. To repair/maintain distributed systems is a complex task, because there are different parts of the system that run in different machines, and could be developed by different groups of people, each group using distinct programming languages. So, the use of tools to help the maintenance of distributed systems is very important and useful.

A good option to help this maintenance is process mining, which can create an explicit process model through the events logged in a file. By this explicit model it would be possible to verify how the system's modules are related during some task, making it easy to find out the faults in the system, and making the maintenance of the system easier.

This work is going to present an approach to do one of the steps of a trace system for middleware, the data collection. The main function of this step is to collect the middleware's activities, sort and format them, creating a system's activities log, which can be used by a process mining tool.

Keywords: Distributed Systems, Middleware, Ordering of Events, Log, Interpolation

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

IoT	*Internet of Things*
IBSG	*Internet Business Solutions Group*
XES	*Extensible Event Stream*
EJML	*Efficient Java Matrix Library*
RTT	*Round-Trip Time*

Sumário

1 Introdução

Neste capítulo serão abordados os principais aspectos deste trabalho, através da introdução do assunto e da importância do tema em que se baseia esta pesquisa. Também serão apresentados os objetivos e a estrutura da monografia.

1.1 Contextualização e motivação

Desde a criação dos computadores, diversas aplicações foram desenvolvidas, e parte destas aplicações marcaram época, fazendo com que a humanidade evoluísse de uma maneira mais ágil.

Inicialmente, os computadores eram utilizados por grandes empresas, não se pensava em utilizar o computador como um objeto pessoal. Desde que o computador começou a ser pensado como tal, a quantidade de aplicações desenvolvidas aumentou e a interação/familiaridade das pessoas com computadores cresceu junto.

Um dos principais motivos para a popularização do computador foi a criação da Internet. A Internet começou a disseminar nos anos 90, porém naquela época ela ainda era bastante limitada por conta de seu baixo desempenho, ou seja, não era possível desenvolver aplicações que exigissem alto desempenho da rede. A partir dos anos 2000 foi que a Internet se tornou essencial para muitas aplicações, as quais passaram a utilizar a rede constantemente em suas atividade (CHASE, 2013).

Atualmente, é difícil encontrar aplicações que não possuem interação com a Internet. Nos dias de hoje, grande parte dos softwares, até mesmo os mais simples, utilizam a Internet para realizar alguma atividade, fazendo com que diversos dispositivos, não só computadores pessoais, mas também pequenos sensores, utilizem a mesma. Essa utilização da Internet, por diferentes tipos de dispositivos, criou um novo conceito, conhecido como Internet das Coisas, do inglês *Internet of Things* (IoT).

A Internet das Coisas surgiu como um conceito para descrever essa grande quantidade de aparelhos tecnológicos ligados à Internet. De acordo com *Internet Business Solutions Group* (IBSG), IoT é simplesmente uma definição para a época em que o número de ‘objetos e coisas’

conectadas à Internet ultrapassou o número de pessoas no planeta. Essa ultrapassagem ocorreu entre 2008 e 2009, e se deu sobretudo por conta dos *smartphones* e *tablets*. Naquele ano o número de dispositivos conectados à Internet chegou a 12.5 bilhões, enquanto a população mundial era de 6.8 bilhões. E esse número só tende a crescer, estudos indicam que em 2020, aproximadamente 50 bilhões de dispositivos estarão conectados a Internet (EVANS, 2011).

Esse grande número de dispositivos ocasionou um grande aumento na produção de novas aplicações tecnológicas nos últimos anos, alavancando a indústria de softwares e fazendo com que, além das simples aplicações, sistemas distribuídos (MISHRA; TRIPATHI, 2014) começassem a ser bastante utilizados, como por exemplo, o Google, sites de compra online ou até mesmo algum jogo para dispositivos móveis.

Tais aplicações, distribuídas ou não, são produzidas e mantidas por diversas empresas, muitas delas multinacionais. E como todo produto, a produção e manutenção de software gera gastos para as empresas. Nestes gastos, a manutenção aparece como maior vilã, pois ela é responsável por mais de 90% do custo de um software (J. BURKI; H. VOGT, 2014). Dentre alguns fatores que justificam este alto custo com a manutenção, pode-se incluir a falta/deficiência de documentação e a substituição de pessoas que possuem mais conhecimento a respeito do software.

Essa manutenção se torna ainda mais custosa em sistemas mais complexos, como os sistemas distribuídos, onde existem vários pedaços de softwares espalhados em diferentes máquinas e que podem ter sido desenvolvidos por diferentes grupos, usando diferentes linguagens de programação (AGUILERA et al., 2003).

Por conta deste alto custo, a utilização de ferramentas que auxiliem a manutenção de sistemas distribuídos se torna cada vez mais útil e necessário.

Uma boa alternativa para auxiliar a manutenção desses sistemas é a mineração de processos (AALST; WEIJTERS, 2004). O objetivo dela é extrair um modelo explícito do processo a partir de eventos registrados em um arquivo de log. Por meio deste modelo é possível verificar como os módulos de um sistema estão se relacionando na realização de determinada tarefa, facilitando, desta forma, a identificação de falhas e, consequentemente, a manutenção do sistema.

1.2 Objetivos

Um importante componente da mineração de processos é o log de atividades da aplicação. Esse log pode ser obtido através de ferramentas de *trace* (SIGELMAN et al., 2010)(FONSECA et al., 2007)(ANDERSON et al., 2009).

Todas essas ferramentas procuram registrar as atividades realizadas por uma aplicação, a fim de facilitar a identificação de um mau funcionamento da mesma, com o objetivo final de corrigir os problemas encontrados, melhorando cada vez mais o funcionamento do sistema como um todo.

Em sistemas distribuídos, o desenvolvimento dessas ferramentas de *trace* deve levar em consideração vários fatores, dentre eles, a baixa sobrecarga imposta no sistema, e um registro ordenado dos eventos ocorridos, já que estes acontecem em diferentes máquinas.

O objetivo deste trabalho é desenvolver uma das etapas, a coleta de dados, presentes um sistema de *trace* para *middleware* (BERNSTEIN, 1996) (VINOSKI, 2002), a qual pode ser observada na Figura 1.1. O *middleware* aparece como uma ferramenta bastante utilizada no desenvolvimento de aplicações distribuídas, pois ele fornece uma camada de software entre as aplicações, o sistema operacional e as camadas de comunicação de rede, abstraindo dificuldades de implementação do sistema, facilitando assim o trabalho do desenvolvedor (RAZZAQUE et al., 2016).

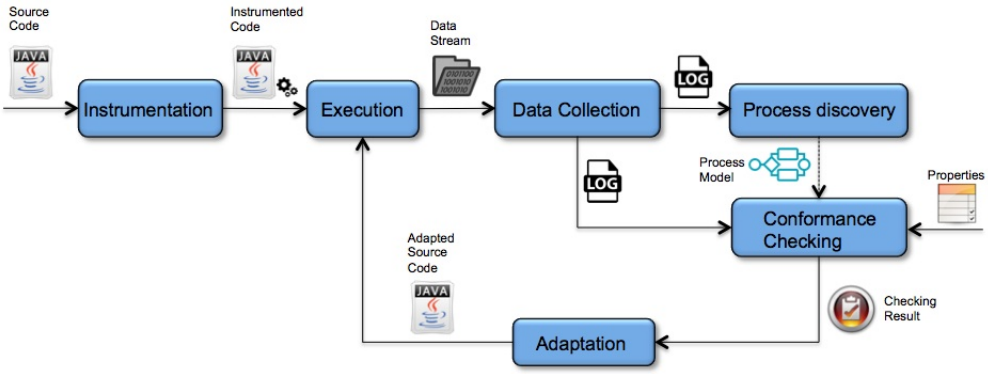

Figura 1.1: Esquema do sistema para fazer mineração de processos em *middleware*

A coleta de dados tem como função principal coletar os eventos realizados pela execução do *middleware*, ordená-los e formatá-los, gerando um log de atividades do sistema para uma posterior análise.

A versão final do log será gerada utilizando o padrão *Extensible Event Stream* (XES). O formato XES é comumente utilizado como um padrão para registro de eventos. Ele possui como algumas de suas principais características uma formatação simples, que pode ser facilmente entendida por qualquer pessoa, e compatibilidade com qualquer domínio/aplicação (XES, 2010).

Este log no padrão XES vai servir como entrada para o *framework* ProM. O ProM é uma ferramenta de mineração de processos, pela qual se extrai informações do sistema através dos eventos registrados no log. As principais vantagens do ProM são sua flexibilidade, pois ele suporta diferentes formatos de arquivo para entrada e saída de dados, e a facilidade que ele proporciona para o desenvolvimento de novas funções, permitindo uma fácil reutilização de seu código (VAN DONGEN et al., 2005).

1.3 Estrutura do trabalho

Este trabalho é composto por 4 capítulos. No Capítulo 2 é discutida a importância do arquivo de log para mineração de processos, as principais dificuldades para geração deste tipo

de arquivo em uma arquitetura de software distribuído, e, por fim, é apresentada a abordagem adotada neste trabalho para a geração do log. No Capítulo 3 são demonstrados os detalhes da implementação para realização da coleta de dados, falando das etapas envolvidas para geração de um arquivo de log final. O Capítulo 4 explica os testes que foram feitos para validar os métodos desenvolvidos, apresentando os resultados obtidos. Por fim, o Capítulo 5 traz as considerações finais, as dificuldades encontradas para realização deste trabalho e possíveis trabalhos futuros.

2

Arquivos de log

Atualmente, a grande maioria dos sistemas, distribuídos ou não, produzem algum arquivo de log . Neste arquivo são registrados diferentes eventos do software, tais como: Chamadas a funções, valores de variáveis passadas por parâmetro, retornos de funções, tempo de início e término das funções, entre outros. O que é salvo no arquivo de log pode variar de sistema para sistema, porém, a utilidade deste arquivo para qualquer aplicação é a mesma, monitoramento e auxílio tanto no desenvolvimento quanto na manutenção do sistema (ANDREWS, 1998).

A utilização destes arquivos de log tem se mostrado algo essencial para grandes aplicações, pois eles ajudam tanto na identificação de problemas do software como na sua depuração (AALST et al., 2006). Através do log de um sistema é possível seguir a lógica do programa, em alto nível de abstração, sem precisar executá-lo em modo de depuração. Por conta desta possibilidade de montar o fluxo de um sistema a partir do log, que ele é parte fundamental na mineração de processos.

Neste capítulo é explicada a importância dos logs e como eles são utilizados na mineração de processos, seguido por uma discussão dos principais problemas para geração de log em sistemas distribuídos, e, por fim, é apresentada qual abordagem foi escolhida para realização da coleta de dados.

2.1 Mineração de processos e a utilização do log

A aplicação da técnica de mineração de processos tem mostrado que, tanto gerentes quanto usuários de um sistema, tendem a superestimar seus conhecimentos sobre os processos nos quais estão envolvidos. Em muitos casos, as decisões tomadas para modificações de um processo, se baseiam em diagramas de *PowerPoint*, documentações e em conhecimentos repassados por pessoas mais experientes, o que acaba dificultando o entendimento do sistema, bem como sua manutenção e melhora (VAN DER AALST, 2011).

A mineração de processos tem como principal objetivo extrair, monitorar e melhorar os processos realizados por uma aplicação, através da extração de informações contidas nos logs de eventos (VAN DER AALST, 2012). Esses eventos representam as atividades realizadas

por um software durante seu funcionamento. Portanto, a mineração de processos se baseia em reais eventos, e não em opiniões ou experiências de diferentes pessoas.Desta forma, a mineração de processos pode ser vista como um raio-x, mostrando o que realmente acontece dentro dos processos executados por uma aplicação qualquer.

O ponto inicial para mineração de processos é o arquivo de log dos eventos. Cada linha deste arquivo corresponde a um evento que descreve a atividade realizada pelo sistema. A Figura 2.1 mostra um trecho de um arquivo de log qualquer, onde cada linha possui a informação de qual atividade foi executada por determinado módulo da aplicação, em um tempo específico.

```
1 ServerRequestHandler;receive;start;89806165514003.000000000000000000
2 NamingProxy;bind;start;89809253985998.000000000000000000
3 Requestor;invoke;start;89809270843178.000000000000000000
4 Marshaller;marshall;start;89809278011856.000000000000000000
5 Marshaller;marshall;complete;89809299984689.000000000000000000
6 ClientRequestHandler;send;start;89809301660899.000000000000000000
7 ClientRequestHandler;send;complete;89809305202360.000000000000000000
8 ServerRequestHandler;receive;complete;89809305492489.000000000000000000
9 ClientRequestHandler;receive;start;89809306890199.000000000000000000
10 Marshaller;unmarshall;start;89809307115245.000000000000000000
```

Figura 2.1: Exemplo de um log de eventos

Este log pode ser usado de diferentes maneiras pela mineração de processos, fazendo com que ela seja classificada em três principais tipos (VAN DER AALST; DUSTDAR, 2012), como pode ser visto na Figura 2.2 .

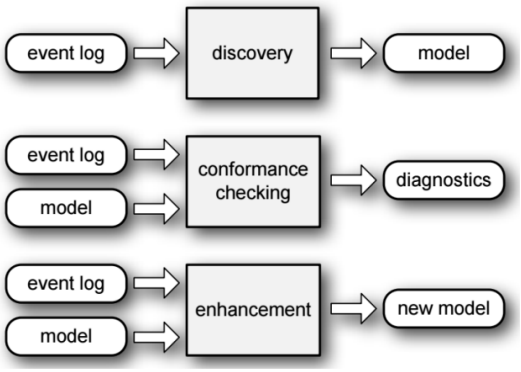

Figura 2.2: Três diferentes tipos de mineração de processos

O primeiro tipo, chamado de *discovery*, recebe um log de eventos como entrada e produz um modelo do processo registrado por este log, sem nenhum conhecimento prévio. Nesta tipo, a única fonte de informação para criação do processo do software é o log.

O segundo tipo tem o nome de *conformance*. Nele, um modelo de processo, previamente definido, é comparado com o log de eventos deste mesmo processo. Este tipo de mineração pretende checar se as atividades descritas no modelo existente, são encontradas, da mesma maneira, no log de eventos, e vice e versa.

O terceiro e último tipo de mineração é chamado de *enhancement*. Aqui, a ideia é estender ou melhorar um processo já existente, utilizando-se das informações presentes no log.

Como pode-se verificar, o log é um elemento fundamental na mineração de processos, e portanto, é necessário ter cuidados durante a sua geração. Na próxima seção são relatadas as principais dificuldades na geração de logs em sistemas distribuídos, que é o foco deste trabalho.

2.2 Geração de log em sistemas distribuídos

Gerar um log de atividades em sistemas não distribuídos apresenta-se como uma simples tarefa de coletar informações durante a execução da aplicação. Mesmo essa simples atividade de registrar os eventos que ocorrem em uma única máquina, gera preocupações em relação ao desempenho do software, pois este registro de atividades deve afetar o mínimo possível o desempenho da aplicação.

Já em sistemas distribuídos, como, por exemplo, o *middleware*, além desta preocupação com o desempenho, existe a dificuldade em ordenar os eventos, pois estes ocorrem em diferentes máquinas, as quais podem está com tempos diferentes, tornando difícil saber se um evento ocorreu antes ou depois do outro.

A seguir são explicadas, com mais detalhes, as preocupações relacionadas com ordenação dos eventos e com o desempenho na geração do log de um sistema distribuído .

2.2.1 Ordem dos eventos

Em sistemas distribuídos não existe um relógio global. Cada máquina que executa alguma parte do sistema possui seu próprio relógio, aumentando a dificuldade para ordenar as atividades que estejam rodando em computadores distintos. Portanto, registrar e ordenar eventos que ocorrem em diversas máquinas torna-se uma tarefa complexa.

Uma simples alternativa seria a utilização do protocolo NTP (MILLS, 1992). O NTP é um protocolo que tem como função sincronizar os relógios físicos de diferentes máquinas. Embora este protocolo seja bastante utilizado, ele possui limitação em relação a granularidade da sincronização dos relógios,não sendo suficientemente preciso, caso seja necessário uma alta precisão nos tempos registrados (VALDMAN, 2001). Por exemplo, funções que executam com uma diferença de tempo muito curta, alguns micro segundos, em computadores diferentes, podem ser registradas como se tivessem sido executadas em um mesmo momento.

Uma outra abordagem bastante comum para ordenação de eventos em software distribuído, é a utilização de um relógio lógico. Esta abordagem é conhecida como ordem parcial dos eventos (LAMPORT, 1978), pois, ao invés de usar o relógio real, como faz o NTP, ela utiliza um tempo lógico, o qual não tem a necessidade de possuir relação com o tempo real (podendo, inclusive, ser um contador), e serve apenas para definir se determinada função aconteceu antes ou depois de uma outra função qualquer, independentemente de onde elas foram executadas. Nesta

abordagem, normalmente, os tempos atribuídos a cada evento não indicam hora/minuto/segundo exato no qual ele foi executado. Nela, o tempo funciona apenas como um identificador, o qual servirá para ordenar os eventos.

2.2.2 Desempenho

Toda atividade de log que é adicionada a uma aplicação vai acarretar em uma perda de desempenho, pois mais tarefas serão executadas durante a atividade do software. Apesar disso, existe um cuidado em fazer com que essa perda seja a menor possível. Algumas alternativas são citadas em Andrews (ANDREWS; COMPUTER SCIENCE, 1998).

Uma alternativa simples, seria incluir todas as funções necessárias para realização da atividade de log na aplicação, fazendo com que elas sempre fossem executadas junto com o software. O maior problema dessa abordagem é o fato de que sistemas de log envolvem constantes e custosas operações de escrita, normalmente em disco rígido, e, geralmente, os arquivos gerados ocupam muito espaço em disco, fazendo com que, por mais que se otimize, o desempenho da aplicação sofra uma diminuição.

Uma segunda alternativa seria fazer com que o sistema de log ficasse separado da aplicação, fazendo com que ele só fosse executado quando necessário, não afetando, desta forma, o sistema de uma forma constante. O problema que esta abordagem pode causar é a diferença de comportamento da aplicação sendo executado com o sistema de log e sem esse sistema.

Uma melhor opção seria um meio termo entre as duas alternativas anteriores. Incluir o sistema de log no código do software, como propõe a primeira alternativa, porém, controlar a execução das funções de log através de uma variável global, ativando e desativando o log sempre que se desejar. Desta forma, a aplicação não vai possuir uma perda de desempenho em todas as suas execuções. O custo que sempre vai existir nesta abordagem é o da escolha entre usar as funções de log ou não durante a execução do programa. Contudo, este chaveamento adiciona pouca carga ao sistema como um todo.

2.2.3 Abordagem escolhida

Tendo em vista os problemas apresentados anteriormente, o primeiro passo para efetuar a coleta de dados, foi a escolha da abordagem a ser seguida. As principais preocupações foram, exatamente, as questões de desempenho e ordenação das atividades do software, discutidas na seção anterior.

A fim de conseguir ordenar eventos que ocorrem em diferentes máquinas, a coleta dos dados foi feita baseada no conceito de ordem parcial dos eventos, o qual procura resolver o problema de ordenação entre as atividades sem se preocupar com o tempo real em que elas foram executadas.

Mesmo dentro desta abordagem de ordem parcial, existem diferentes estratégias que podem ser adotadas. Neste trabalho foi utilizado o procedimento apresentado em Anderson et

al. (ANDERSON et al., 2009). A ideia do procedimento é utilizar uma ferramenta matemática (interpolação linear em trechos (HILDEBRAND, 1987)) para ordenar os eventos.

A interpolação é um método matemático que permite a construção de um conjunto de dados a partir de um outro conjunto discreto de dados pontuais previamente conhecidos.

Para exemplificar o funcionamento da interpolação, suponha que se tenha um conjunto de dados, previamente conhecidos, representados pelos pontos da Figura 2.3.

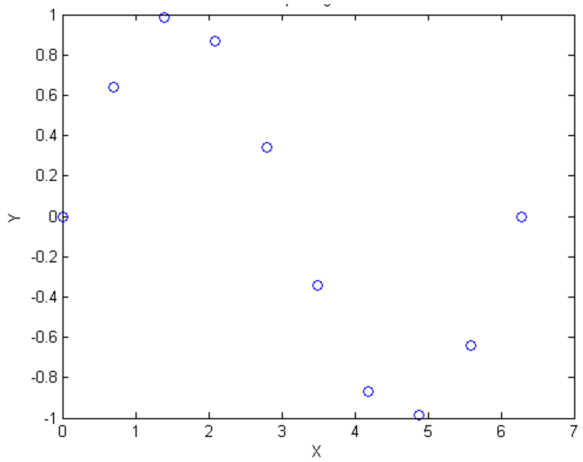

Figura 2.3: Exemplo de pontos utilizados pela função de interpolação

Esses pontos possuem as seguintes coordenadas:

Tabela 2.1: Coordenadas dos pontos utilizados na interpolação

Valores no eixo X	Valores no eixo Y
0	0
0.698131700797732	0.642787609686539
1.39626340159546	0.984807753012208
2.0943951023932	0.866025403784439
2.79252680319093	0.342020143325669
3.49065850398866	-0.342020143325669
4.18879020478639	-0.866025403784438
4.88692190558412	-0.984807753012208
5.58505360638185	-0.64278760968654
6.28318530717959	-2.44929359829471e-16

Embora só se tenham 10 pontos, através da interpolação é possível calcular, de maneira aproximada, o valor y de qualquer ponto x' que esteja dentro do intervalo de pontos x previamente conhecidos, apresentados na tabela acima. A Figura 2.4 mostra o ponto, representado pelo 'x', gerado pela interpolação para o valor de x = 1.3.

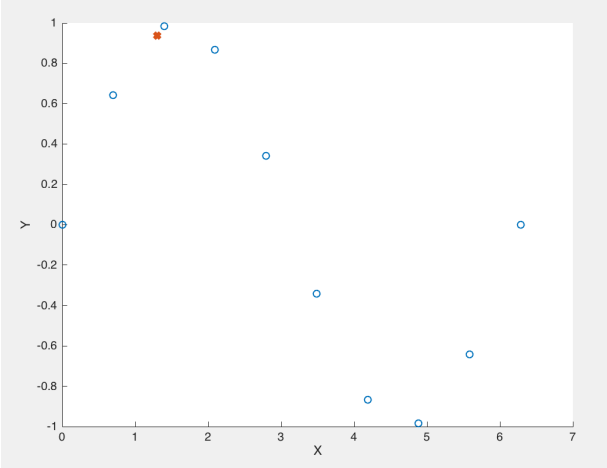

Figura 2.4: Resultado Interpolação para x = 1.3

Seguindo esta ideia de gerar valores para pontos desconhecidos, a função de interpolação em trechos pode ser utilizada para gerar valores y's de infinitos pontos dentro de um intervalo de valores x's específicos, tornando possível a criação do gráfico presente na Figura 2.5.

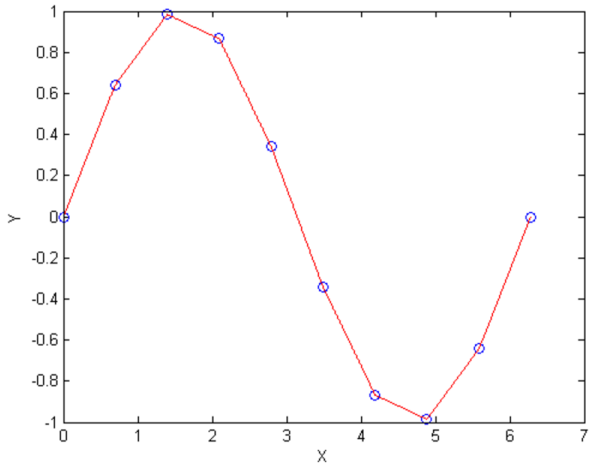

Figura 2.5: Exemplo de resultado da interpolação em trechos

Na coleta de dados, a interpolação irá trabalhar com pares x,y correspondentes ao tempo do cliente e do servidor, respectivamente. Todo tempo x registrado no cliente terá um tempo y correspondente no servidor. Através deste procedimento é possível unificar os tempos dos eventos que acontecem nas diferentes máquinas e ordená-los.

No quesito referente ao desempenho, a coleta de dados utiliza a abordagem híbrida. Ela foi desenvolvida de tal forma que possa ser incluída no código de uma aplicação, a qual pode escolher utilizá-la ou não em tempo de execução, através de uma variável de controle.

No próximo Capítulo 3 serão mostrados os detalhes da implementação da coleta de dados, apresentando como a interpolação é utilizada e quais os principais problemas relacionados ao desempenho que esta abordagem pode causar.

3

Coleta de dados

A coleta de dados é uma das etapas presentes em um sistema de *trace* para *middleware*. As tarefas realizadas durante esta etapa foram desenvolvidas utilizando-se a linguagem de programação Java, e elas têm como função principal gerar um log ordenado de atividades executadas pelo *middleware*.

A coleta de dados funciona da seguinte maneira: Recebe-se, por meio de pacotes de rede, as informações de eventos executados pelas diferentes partes do *middleware*, e gera-se, como resultado, um arquivo único e ordenado desses eventos. Portanto, a coleta dos dados é feita em uma arquitetura de cliente-servidor, onde a máquina responsável por esta coleta desempenha o papel de servidor e as outras diversas máquinas onde o *middleware* está sendo executado desempenham o papel de clientes, os quais enviam informações de log diretamente para o servidor.

Nesta capítulo é explicado como é feita a coleta dos dados, abordando detalhes da implementação da abordagem escolhida para sua realização, abordagem esta que foi apresentada no Capítulo 2.

3.1 Detalhes da implementação

A máquina responsável pela coleta dos dados funciona como um servidor de logs. Cada parte do *middleware*, sempre que desejar registrar alguma atividade, ao invés de escrever essa informação em um arquivo local, deve enviar todas as informações necessárias para o servidor através da rede. Esse envio é feito por meio de uma chamada de função assíncrona, ou seja, chama-se a função para o envio dos dados e continua-se o fluxo normal do programa, não é preciso esperar o envio da mensagem ser finalizado.

Os dados enviados ao servidor são salvos em arquivos de texto simples, e cada cliente possui seu próprio arquivo de texto no servidor, também chamado de arquivo de log. Dentre os dados enviados, está o tempo em que a atividade foi executada no cliente. Este tempo é chamado de *timestamp*, e é utilizado para ordenação das atividades do *middleware*. Como os *timestamps* são originados de diferentes máquinas, não é possível compará-los da forma que eles chegam.

Para torná-los comparáveis, utiliza-se a técnica citada no Capítulo 2, a interpolação.

O primeiro passo para realizar a interpolação é fazer coletas constantes dos tempos das diversas máquinas onde está sendo executado o *middleware*. Estas coletas de tempos são necessárias para poder calcular a diferença entre os tempos dos computadores em determinado momento.

A coleta dos tempos também funciona em uma arquitetura cliente-servidor, sendo executada em paralelo com a atividade de registro das ações. Cada cliente se conecta com o servidor e, periodicamente, troca informações (chamadas de *echo*) relativas ao seu *timestamp*. A imagem Figura 3.1 apresenta um diagrama das mensagens trocadas entre o cliente (partes do *middleware*) e o servidor (máquina responsável pela coleta dos dados).

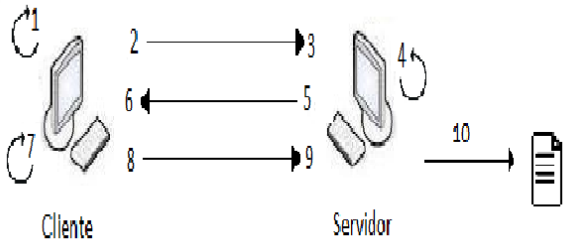

Figura 3.1: Troca de mensagens entre as máquinas para realização da coleta de tempos

O processo apresentado acima funciona da seguinte forma:

1. O cliente registra seu atual *timestamp* (*sendTime*);
2. O cliente envia mensagem contendo o tempo registrado em 1 para o servidor;
3. O servidor recebe a mensagem;
4. O servidor registra seu *timestamp* (*remoteTime*);
5. O servidor envia a resposta com o seu *timestamp* adicionado para o cliente;
6. O cliente recebe a resposta do servidor;
7. O cliente registra seu tempo mais uma vez (*replyTime*);
8. O cliente reenvia a mensagem ao servidor;
9. O servidor recebe a mensagem final contendo os três *timestamps* (*senTime*, *remoteTime* e *replyTime*);
10. O servidor salva os dados contidos no pacote de rede (*senTime*, *remoteTime* e *replyTime*) em um arquivo de texto específico, relacionado ao cliente.

Através desta troca de informações, é possível calcular os seguintes valores:

1. *Round-Trip Time* (RTT).;
2. *LocalTime*;
3. *Skew*.

O RTT é o tempo que a mensagem demora para chegar no servidor mais o tempo que a resposta do servidor gasta para chegar no cliente. Ele pode ser calculado da seguinte forma: RTT = *replyTime* - *sendTime*.

O *lacalTime* é o tempo estimado no qual o servidor enviou a resposta para o cliente. Este tempo é estimado a partir do relógio do cliente, e ele baseia-se na ideia de que o atraso na rede é simétrico, ou seja, o tempo gasto para uma mensagem ir de A até B é o mesmo tempo gasto para esta mensagem ir de B até A. Como esta afirmação nem sempre é verdadeira, o valor torna-se uma aproximação. Ele pode ser calculado da seguinte maneira *localTime* = *sendTime* + RTT/2.

O *skew* é a estimada diferença entre os tempos do cliente e do servidor. Ele é obtido através do *localTime*, fazendo-se a seguinte operação: *skew* = *remoteTime* - *localTime*. Por conta do *localTime* ser uma aproximação, o resultado encontrado para o *skew* também é uma aproximação.

Todos estes valores coletados e calculados são úteis para a realização da interpolação. Mais precisamente, três destes valores (*sendTime*, *skew* e *rtt*) são diretamente utilizados. Esses valores são salvos em arquivos de texto simples, para serem usados, posteriormente, na interpolação propriamente dita. A Figura 3.2 mostra um exemplo do arquivo de texto com os valores salvos, na primeira coluna fica o *sendTime*, na segunda o *skew* e na terceira o *rtt*.

```
SendTime                                Skew                                RTT
1 246194374260840.000000000000000000;8476189.000000000000000000;17725738.000000000000000000
2 246194393365438.000000000000000000;8719679.000000000000000000;17740012.000000000000000000
3 246194412222437.000000000000000000;3145873.500000000000000000;6474457.000000000000000000
4 246194419807059.000000000000000000;46620.000000000000000000;250358.000000000000000000
5 246194421302646.000000000000000000;31317.500000000000000000;245969.000000000000000000
6 246194422790625.000000000000000000;29580.500000000000000000;255741.000000000000000000
7 246194424379132.000000000000000000;11563.500000000000000000;201543.000000000000000000
8 246194425917917.000000000000000000;8306.500000000000000000;236699.000000000000000000
9 246194427508161.000000000000000000;13145.500000000000000000;205969.000000000000000000
10 246194428830250.000000000000000000;2455.000000000000000000;244318.000000000000000000
```

Figura 3.2: Arquivo de texto gerado pela coleta de tempos

O RTT é utilizado como um filtro para eliminar os *outliers* presentes no conjunto de dados. *Outliers* são valores que destoam dos dados presentes em um conjunto qualquer. Eles representam valores que, por algum motivo externo, se comportaram de uma maneira muito diferente durante a coleta dos dados. No caso da coleta dos tempos, um *outlier* pode ocorrer durante uma sobrecarga na rede, o que acaba gerando atraso no envio dos pacotes, aumentando o RTT.

Este filtro identifica os *outliers* a partir do *box plot* montado com os valores do RTT. Esta identificação de *outliers* é apresentada em (LAURIKKALA et al., 2000), e nesta abordagem, todo ponto que possui valor menor que o adjacente superior ou maior que o adjacente superior são considerados *outliers*, também chamados de discrepantes, como mostra a Figura 4.1. O adjacente superior é calculado da seguinte forma: (terceiro quartil + (1.5 * (valor do terceiro quartil - valor do primeiro quartil))), enquanto o adjacente inferior é obtido de outra maneira: (primeiro quartil + (1.5 * (valor do terceiro quartil - valor do primeiro quartil))).

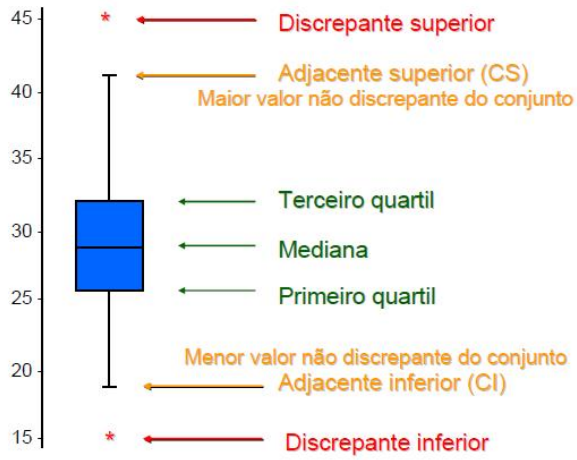

Figura 3.3: Exemplo de boxplot

Após a realização da filtragem, é chamada a função de interpolação. De todos os valores obtidos/calculados, ela utiliza dois deles (*sendTime* e *skew*) diretamente. O *sendTime* e o (*sendTime* + skew) representam, respectivamente, um par (x,y), onde o x (*sendTime*) representa o tempo do cliente, e o y (*sendTime* + *skew*) representa o valor do relógio do cliente de acordo com tempo do servidor.

Como a coleta de tempos acontece constantemente, um conjunto de pares (x,y) são gerados. Este conjunto, ordenado de acordo com os valores x do par (x,y), junto com um outro conjunto de pontos z, correspondente aos tempos registrados nos arquivos de log, são passados como parâmetros de entrada para a função de interpolação, a qual gera como saída, um conjunto de pontos y', que representa os valores y's relativos aos valores x's presentes no conjunto z de entrada.

Como cada computador possui seu próprio arquivo de log no servidor, a função de interpolação é aplicada em cada um dos arquivos, unificando os *timestamps* das diferentes máquinas.

Essa unificação dos tempos torna possível a comparação entre os *timestamps* dos diferentes clientes, pois ela transforma um tempo tx, de um cliente sendo executado em qualquer máquina, para um tempo ty, correspondente ao relógio do servidor, ou seja, todos os tempos transformam-se em valores relativos a um único relógio, tornando-os comparáveis. Ao final da utilização da interpolação em cada arquivo de log, os novos tempos obtidos são comparados,

gerando um arquivo de log único e ordenado.

Este arquivo de log unificado é gerado em formato de texto e com valores de tempos em nanosegundos. A fim de permitir a utilização do log para mineração de processos através do ProM, faz-se necessário transformar o arquivo de texto para o formato XES. O XES é um formato de arquivo baseado em XML, ou seja, todos os dados obtidos no log são organizados em diferentes *tags* dentro do arquivo. A única mudança, feita diretamente nos dados do log, está relacionada com os *timestamps*, os quais não representam o tempo real da execução de um evento, como já foi dito anteriormente, e só servem para indicar a ordem dos eventos. Portanto, tais tempos podem ser alterados para quaisquer valores, desde que a ordenação dos eventos seja preservada. Como o formato XES exige que os *timestamps* estejam no formato '2016-11-24T00:10:46.188-0300', o qual possui precisão máxima de milisegundos, foi preciso, ao invés de utilizar o tempo presente no log, transformar o tempo de cada linha para uma data qualquer no formato exigido pelo padrão XES. As linhas são transformadas em ordem, e em cada linha é adicionado 10 minutos ao tempo, para manter a ordem dos eventos. Logo, ao final do processo de transformação, os tempos dos eventos que aparecem em sequência possuem diferença de 10 minutos entre eles, o que preserva a ordem dos eventos, sem se importar com o real tempo de execução destes. A tabela 3.1 mostra os tempos antes da transformação e os tempos depois da transformação. Pode-se observar que não existe uma relação lógica entre os tempos da esquerda e os tempos da direita, a única condição é de que a ordem dos tempos após a transformação preserve a ordem dos tempos antes da transformação

Tabela 3.1: Transformação dos tempos para o formato XES

Valores antes da transformação	Valores depois da transformação
276591956993474.000000000000000000	2016-11-24T00:10:46.188-0300
276594340229138.030000000000000000	2016-11-24T00:20:46.188-0300
276594361741448.250000000000000000	2016-11-24T00:30:46.188-0300
276594372684248.300000000000000000	2016-11-24T00:40:46.188-0300
76594395448235.300000000000000000	2016-11-24T00:50:46.188-0300
276594396901485.750000000000000000	2016-11-24T01:00:46.188-0300
276594407380529.160000000000000000	2016-11-24T01:10:46.188-0300
276594407851879.840000000000000000	2016-11-24T01:20:46.188-0300
276594409204110.750000000000000000	2016-11-24T01:30:46.188-0300
276594409564816.700000000000000000	2016-11-24T01:40:46.188-0300

Além desta preocupação com a transformação dos *timestamps*, é preciso ter atenção com o desempenho das tarefas realizadas. A coleta e unificação dos tempos, e a geração do arquivo XES trabalham com uma grande quantidade de valores, sendo necessário tomar cuidados em relação ao desempenho da aplicação como um todo, a fim de afetar minimamente o desempenho do sistema.

Dentre as atividades presentes durante a coleta dos dados, três fatores precisam ser levados em consideração: O fluxo de pacotes gerado pela coleta dos tempos, o uso de espaço em

disco para salvar os logs e as informações da coleta, e a manipulação dessas informações para geração de um log ordenado e unificado.

Embora a coleta dos tempos seja executada constantemente junto com as outras atividades do sistema, ela não afeta a aplicação de uma forma preocupante, pois o tráfego gerado por ela não sobrecarrega a rede, já que os dados trocados entre as máquinas, por conta da coleta, são poucos, apenas pacotes com o conteúdo de, no máximo, três valores numéricos, relativos aos tempos registrados.

Como cada máquina possui seu próprio arquivo com dados da coleta e seu próprio arquivo de log, ambos salvos no servidor, a quantidade de dados que necessitam ser armazenados no servidor pode ser muito grande. Tais arquivos, além de ocupar muito espaço em disco rígido, necessitam de um grande espaço de memória, e maior poder de processamento para gerarem o resultado final.

A solução utilizada para minimizar este problema foi fazer uma limpeza, regularmente, dos arquivos que não são mais necessários. Os arquivos deixam de ser úteis a partir do momento que a interpolação utiliza os valores presentes neles. Neste caso, de tempos em tempos, chama-se a função de interpolação para gerar um arquivo único de log e se remove os dados da coleta e os logs individuais gerados até aquele momento.

4

Testes realizados

O Capítulo 3 mostrou detalhes da implementação da coleta dos dados, discutindo como ela trabalha para gerar um log de atividades único e ordenado. Neste capítulo serão apresentados os testes realizados para testar a sistema de coleta e seus resultados.

A principal parte da coleta de dados, a qual foi toda desenvolvida em Java, é a função de interpolação, a qual é responsável por unificar os tempos das diferentes máquinas, tornando-os comparáveis. Portanto, é necessário, antes de testar o funcionamento da coleta em si, fazer a validação da função de interpolação, verificando se esta gera resultados satisfatórios.

Pelo fato da não existência de vários computadores, todos os testes apresentados a seguir foram realizados em um único computador.

4.1 Validação da função de interpolação

Como já foi citado anteriormente, a coleta de dados foi desenvolvida utilizando-se a linguagem de programação Java. Desta forma, foi necessário desenvolver uma função de interpolação nesta linguagem. Tal função foi baseada na função *interp1* do Matlab (MATHWORKS, 2016a), e as operações realizadas por ela foram implementadas com a ajuda da biblioteca *Efficient Java Matrix Library* (EJML) (EJML, 2016).

A fim de validar a função de interpolação, foram realizados dois testes: O primeiro teste procura validar os resultados gerados pela função de interpolação existente no Matlab, e o segundo teste compara a função de interpolação desenvolvida em Java com a função de interpolação do Matlab.

Para realizar o primeiro teste, executou-se a coleta de tempos durante um minuto, e utilizou-se o log gerado por ela. A finalidade deste teste é verificar se os tempos gerados pela função de interpolação preservam a ordem dos eventos ocorridos.

O log da coleta de tempos possui 19855 linhas, cada linha contém um resultado de uma coleta com o tempo do cliente (*sendTime*), a diferença entre o tempo do cliente e o tempo do servidor (*skew*), e o RTT, como já foi mostrado no Capítulo 3. Este log foi dividido, aleatoriamente, em dois conjuntos, chamados de treinamento (14981 linhas) e teste (4874 linhas).

A única restrição para geração dos conjuntos foi de que os tempos do cliente, presentes no conjunto de teste, estivessem compreendidos no intervalo de tempo do cliente presente no conjunto de treinamento.

Após esta divisão executou-se a função *interp1* do Matlab passando como parâmetro o tempo registrado no cliente (*sendTime*), o tempo do cliente de acordo com o relógio do servidor (*sendTime* + *skew*) e o e os valores do tempo do cliente presentes no conjunto de teste. Os valores retornados pela função foram comparados com os valores do tempo do servidor, calculado pelo cliente, obtidos a partir do conjunto de teste.

Esta comparação foi feita através de um teste de hipótese (VARGHA; DELANEY, 1998), utilizando-se a função kruskalwallis (MATHWORKS, 2016b) do MatLab, com nível de significância de 0.01. Neste teste, o p-valor obtido foi de 0.9982, a partir do qual podemos concluir que as duas amostras (valores do *interp1* e do conjunto de testes) pertencem à uma mesma distribuição. A Figura 4.1 mostra o *box plot* para cada conjunto, onde o conjunto 1 representa os valores da função *interp1* e o conjunto 2 os valores do conjunto de teste.

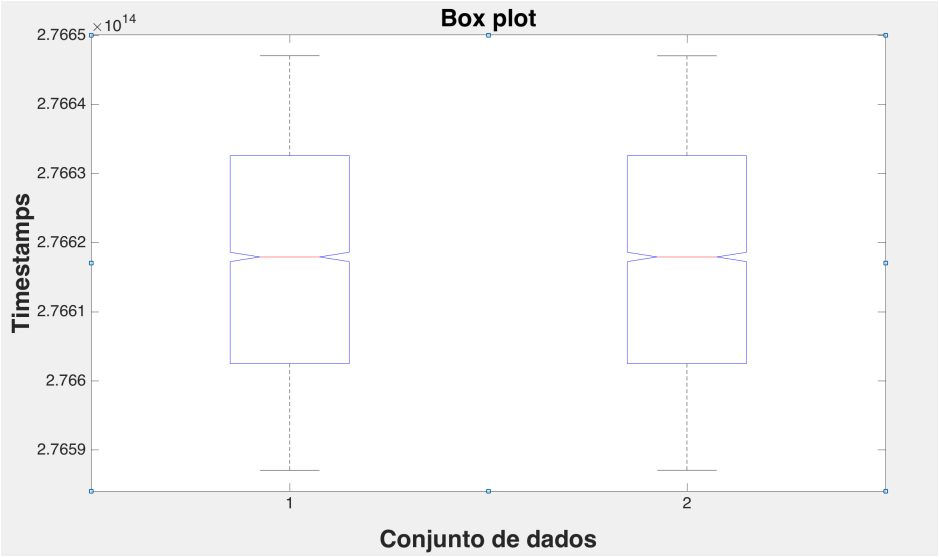

Figura 4.1: *Box plot* da função *interp1* e do conjunto de testes

Os valores gerados pela função *interp1* tiveram pequenas diferenças para os valores reais presentes no arquivo de teste, e, sobretudo, a ordem dos eventos registrados não foi alterada, ou seja, a função de interpolação consegue produzir resultados que preservam a ordem de execução das tarefas realizadas.

Tendo verificado que a função *interp1* gera resultados válidos, é necessário fazer a comparação da função *interp1* desenvolvida em Java com a função *interp1* do Matlab, a fim de verificar equivalência entre as funções.

Para verificar esta equivalência executou-se cada função com as mesmas entradas do teste anterior e coletou-se dados estatísticos do resultado de cada função. Estes dados foram iguais para as duas funções, como pode ser observado na Tabela 4.1.

Tabela 4.1: Dados estatísticos das funções de interpolação

	Interp1 Java	*Interp1* Matlab
Mínimo	276587070760057	276587070760057
Máximo	276647049810022	276647049810022
Média	276617546538803	276617546538803
Mediana	276617945341446	276617945341446
Desvio Padrão	17304740307.6950	17304740307.6950

A partir da verificação dos dados estatísticos ficou claro que a função desenvolvida em Java apresenta o mesmo comportamento da função em Matlab.

4.2 Testando a coleta de dados

Após a validação da função de interpolação desenvolvida em Java, foi realizado um teste da coleta de dados integrada com um software simples.

Executou-se a aplicação, a qual registrou alguns eventos em um arquivo de log, junto com a coleta dos dados, e em um determinado momento foi acionada a interpolação. A interpolação foi responsável por unificar os tempos registrados no log da aplicação. Neste teste procurou-se analisar o resultado final do log gerado pela interpolação.

A Figura 4.2 mostra uma parte do arquivo de log gerado pela aplicação. Utilizando os *timestamps* presentes neste arquivo e os dados obtidos na coleta dos tempos, a função de interpolação transformou os tempos presentes no log da aplicação em tempos relativos ao relógio da máquina responsável pela coleta dos dados, resultando em um novo arquivo, com os tempos modificados, como mostra a Figura 4.3.

```
1  ServerRequestHandler;receive;start;276591956993474.000000000000000000
2  NamingProxy;bind;start;276594340229138.030000000000000000
3  Requestor;invoke;start;276594361741448.250000000000000000
4  Marshaller;marshall;start;276594372684248.300000000000000000
5  Marshaller;marshall;complete;276594395448235.300000000000000000
6  ClientRequestHandler;send;start;276594396901485.750000000000000000
7  ClientRequestHandler;send;complete;276594407380529.160000000000000000
8  ServerRequestHandler;receive;complete;276594407851879.840000000000000000
9  Marshaller;unmarshall;start;276594409204110.750000000000000000
10 ClientRequestHandler;receive;start;276594409564816.700000000000000000
11 Marshaller;unmarshall;complete;276594436972442.470000000000000000
12 NamingImpl;bind;start;276594440426914.440000000000000000
13 NamingImpl;bind;complete;276594442217912.120000000000000000
14 Marshaller;marshall;start;276594446671988.880000000000000000
15 Marshaller;marshall;complete;276594454216623.600000000000000000
```

Figura 4.2: Log gerado pela aplicação utilizada no teste da coleta dos dados

```
ServerRequestHandler;receive;start;276591955806296.000000000000000000
NamingProxy;bind;start;276594338261988.000000000000000000
Requestor;invoke;start;276594359624642.000000000000000000
Marshaller;marshall;start;276594370227748.000000000000000000
Marshaller;marshall;complete;276594395426752.000000000000000000
ClientRequestHandler;send;start;276594396869220.000000000000000000
ClientRequestHandler;send;complete;276594404342561.000000000000000000
ServerRequestHandler;receive;complete;276594404997028.000000000000000000
Marshaller;unmarshall;start;276594407264311.000000000000000000
ClientRequestHandler;receive;start;276594407869106.000000000000000000
Marshaller;unmarshall;complete;276594434616004.000000000000000000
NamingImpl;bind;start;276594439920454.000000000000000000
NamingImpl;bind;complete;276594442191009.000000000000000000
Marshaller;marshall;start;276594444088770.000000000000000000
Marshaller;marshall;complete;276594452852269.000000000000000000
```

Figura 4.3: Log gerado pela aplicação após a transformação dos tempos pela função de interpolação

Após esta transformação dos tempos do log da aplicação, através da interpolação, ocorre a conversão dos tempos de acordo com o formato XES, gerando um novo arquivo com os tempos formatados, como pode ser visto na Figura 4.4.

```
ServerRequestHandler;receive;start;2016-11-24T00:10:46.188-0300
NamingProxy;bind;start;2016-11-24T00:20:46.188-0300
Requestor;invoke;start;2016-11-24T00:30:46.188-0300
Marshaller;marshall;start;2016-11-24T00:40:46.188-0300
Marshaller;marshall;complete;2016-11-24T00:50:46.188-0300
ClientRequestHandler;send;start;2016-11-24T01:00:46.188-0300
ClientRequestHandler;send;complete;2016-11-24T01:10:46.188-0300
ServerRequestHandler;receive;complete;2016-11-24T01:20:46.188-0300
Marshaller;unmarshall;start;2016-11-24T01:30:46.188-0300
ClientRequestHandler;receive;start;2016-11-24T01:40:46.188-0300
Marshaller;unmarshall;complete;2016-11-24T01:50:46.188-0300
NamingImpl;bind;start;2016-11-24T02:00:46.188-0300
NamingImpl;bind;complete;2016-11-24T02:10:46.188-0300
Marshaller;marshall;start;2016-11-24T02:20:46.188-0300
Marshaller;marshall;complete;2016-11-24T02:30:46.188-0300
```

Figura 4.4: Resultado da transformação dos tempos para o formato exigido pelo padrão XES

Este novo arquivo é então convertido para o formato XES, permitindo uma posterior análise da aplicação através de ferramentas de mineração de processos, como o ProM.

5

Conclusão

Neste trabalho foi apresentada uma abordagem para geração de arquivo de log para sistemas distribuídos, explicando os detalhes para seu desenvolvimento, e apresentando os resultados obtidos nos testes realizados. Inicialmente mostrou-se a importância dos arquivos de log para qualquer tipo de aplicação, pois através dele é possível registrar as atividades de um sistema, permitindo, através de uma posterior análise, o descobrimento de falhas. Depois, discutiu-se a importância deste tipo de arquivo na mineração de processos, pois, é através dos dados presentes no log, que a mineração de processos consegue ajudar na descoberta e solução de problemas nas atividades de um sistema. Por fim, apresentou-se o sistema e os resultados dos testes realizados para validação do mesmo.

Pode-se concluir que, embora a tarefa de geração de log em sistemas distribuídos seja bastante complexa, a abordagem aqui escolhida, apresenta-se como uma boa alternativa para ordenação das atividades desse tipo de sistema.

Fica claro que a implementação apresentada neste trabalha não pode ser utilizada para identificar tempos de execução dos eventos em um sistema, pois ela não se importa com o tempo real das atividades executadas, a única preocupação abordada pelo método escolhido é a ordenação de eventos.

Portanto, a utilização da coleta de dados aqui apresentada, só faz sentido caso se queira identificar problemas na ordem das atividades executadas por um sistema, gargalos de tarefas com grande tempo de execução não são abordados pelo log final gerado por este módulo.

5.1 Limitações e dificuldades do trabalho

A principal limitação deste trabalho está na falta de computadores para realização de testes da coleta dos dados com softwares executando em diferentes máquinas. Isto fez com que todos os testes fossem realizados em um único computador, retirando as possibilidades de grandes atrasos durante a troca de informações, fato que pode influenciar nos tempos calculados pela coleta dos dados.

Embora o fato de não existirem outras máquinas seja importante, foi possível verificar que a interpolação, ferramenta matemática utilizada neste trabalho, pode ajudar bastante na complexa tarefa que é a ordenação de eventos que acontecem em aplicações distribuídas. Através dela, a unificação dos tempos das diferentes máquinas de um sistema distribuído torna-se mais precisa do que a obtida no protocolo NTP. Portanto, a interpolação apresenta-se como uma boa alternativa para sistemas distribuídos que precisam sincronizar os relógios das diferentes máquinas onde suas partes são executadas.

5.2 Sugestões para trabalhos futuros

Como trabalho futuro, seria útil aprimorar a função de interpolação para fazer a unificação dos tempos baseado em um relógio real, ou seja, aumentar a quantidade de informações providas pelo arquivo de log, pois, desta forma, seria possível identificar eventos que demoram muito para serem executados, fazendo com que, os responsáveis pelo sistema conseguissem, de forma mais ágil, promover melhoras na execução das atividades do mesmo.

Referências

AALST, W. M. van der et al. Using process mining to analyze and improve process flexibility (position paper). , [S.l.], 2006.

AALST, W. M. Van der; WEIJTERS, A. Process mining: a research agenda. **Computers in industry**, [S.l.], v.53, n.3, p.231–244, 2004.

AGUILERA, M. K. et al. Performance debugging for distributed systems of black boxes. **ACM SIGOPS Operating Systems Review**, [S.l.], v.37, n.5, p.74–89, 2003.

ANDERSON, E. et al. Efficient tracing and performance analysis for large distributed systems. In: IEEE INTERNATIONAL SYMPOSIUM ON MODELING, ANALYSIS & SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS, 2009. **Anais...** [S.l.: s.n.], 2009. p.1–10.

ANDREWS, J. H. Testing using log file analysis: tools, methods, and issues. In: AUTOMATED SOFTWARE ENGINEERING, 1998. PROCEEDINGS. 13TH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 1998. p.157–166.

ANDREWS, J. H.; COMPUTER SCIENCE, U. of Western Ontario. Dept. of. **Theory and practice of log file analysis**. [S.l.]: Citeseer, 1998.

BERNSTEIN, P. A. Middleware: a model for distributed system services. **Communications of the ACM**, [S.l.], v.39, n.2, p.86–98, 1996.

CHASE, J. **The Evolution of The Internet of Things**. [S.l.]: Texas Instruments, 2013.

EJML. **EJML**. Acessado em 20 de novembro de 2016, `http://ejml.org/wiki/index.php?title=Main_Page`.

EVANS, D. **The Internet of Things - How the Next Evolution of the Internet Is Changing Everything** . [S.l.]: Cisco, 2011.

FONSECA, R. et al. X-trace: a pervasive network tracing framework. In: USENIX CONFERENCE ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION, 4. **Proceedings...** [S.l.: s.n.], 2007. p.20–20.

HILDEBRAND, F. B. **Introduction to numerical analysis**. [S.l.]: Courier Corporation, 1987.

J. BURKI, C.; H. VOGT, H. **How to save on software maintenance costs**. [S.l.]: Omnext, 2014.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. **Communications of the ACM**, [S.l.], v.21, n.7, p.558–565, 1978.

LAURIKKALA, J. et al. Informal identification of outliers in medical data. In: FIFTH INTERNATIONAL WORKSHOP ON INTELLIGENT DATA ANALYSIS IN MEDICINE AND PHARMACOLOGY. **Anais...** [S.l.: s.n.], 2000. p.20–24.

MATHWORKS. **interp1**. Acessado em 20 de novembro de 2016, `https://www.mathworks.com/help/matlab/ref/interp1.html`.

MATHWORKS. **kruskalwallis**. Acessado em 20 de novembro de 2016, https://www.mathworks.com/help/stats/kruskalwallis.html.

MILLS, D. Network Time Protocol (Version 3) specification, implementation and analysis. , [S.l.], 1992.

MISHRA, K. S.; TRIPATHI, A. K. Some Issues, Challenges and Problems of Distributed Software System. **International Journal of Computer Science and Information Technologies. Varanasi, India**, [S.l.], v.7, p.3, 2014.

RAZZAQUE, M. A. et al. Middleware for internet of things: a survey. **IEEE Internet of Things Journal**, [S.l.], v.3, n.1, p.70–95, 2016.

SIGELMAN, B. H. et al. Dapper, a large-scale distributed systems tracing infrastructure. , [S.l.], 2010.

VALDMAN, J. Log file analysis. **Department of Computer Science and Engineering (FAV UWB)., Tech. Rep. DCSE/TR-2001-04**, [S.l.], 2001.

VAN DER AALST, W. Process Mining: x-ray your business processes. , [S.l.], 2011.

VAN DER AALST, W. Process mining: overview and opportunities. **ACM Transactions on Management Information Systems (TMIS)**, [S.l.], v.3, n.2, p.7, 2012.

VAN DER AALST, W. M.; DUSTDAR, S. Process mining put into context. **IEEE Internet Computing**, [S.l.], v.16, n.1, p.82–86, 2012.

VAN DONGEN, B. F. et al. The ProM framework: a new era in process mining tool support. In: INTERNATIONAL CONFERENCE ON APPLICATION AND THEORY OF PETRI NETS. **Anais...** [S.l.: s.n.], 2005. p.444–454.

VARGHA, A.; DELANEY, H. D. The Kruskal-Wallis test and stochastic homogeneity. **Journal of Educational and Behavioral Statistics**, [S.l.], v.23, n.2, p.170–192, 1998.

VINOSKI, S. Where is Middleware. **IEEE Internet Computing**, [S.l.], v.6, n.2, p.83–85, Mar 2002.

XES. **Extensible Event Stream**. Acessado em 10 de setembro de 2016, http://www.xes-standard.org/.