1
Introdução

Atualmente, desenvolvimento de software ainda é um esforço principalmente laboral, e por isso, sujeito as limitações humanas [1]. Quando sistemas de software apresentam um funcionamento não esperado, é provável que a identificação da causa raiz do mau funcionamento seja igualmente laboriosa, dependendo de paciência e atenção, principalmente devido a crescente complexidade desses sistemas.

O comportamento esperado de um software é que este siga as especificações definidas em seu projeto. Assim, se houver qualquer comportamento, do modelo do software ou em sua programação, que acarrete em uma não conformidade do que foi especificado, gerando um resultado não esperado, é considerado um defeito do software ou *bug*. Nesse sentido, segundo Gray [2], os defeitos podem ser caracterizados como:

- **Repetitivos:** Sempre que se executa o software, o resultado não corresponde à especificação.
- **Degradação de Desempenho:** O software funciona como especificado, porém com tempo de serviço elevado.
- **Intermitentes:** Ao executar o software, ocasionalmente, o resultado não corresponde à especificação.

Os defeitos repetitivos, ou *bohrbugs* [2], são relativamente fáceis de serem reproduzidos, pois dada uma entrada ao software que gere um defeito, sempre esta mesma entrada ocasionará o mesmo defeito. Os defeitos de degradação de desempenho, entretanto, necessitam de mais esforço para serem reproduzidos, por ser necessário estimular certa quantidade de carga de processamento no software. Os de mais difícil reprodução são os defeitos intermitentes, também conhecidos como *heisenbugs* [2], por depender principalmente da ordem de escalonamento, rede ou sequência de resposta de E/S assíncrona.

Os processos que compõe um software distribuído são comumente executados em máquinas distintas, logo, *bohrbugs*, *heisenbugs* e problemas de desempenho ganham novas dimensões de localização, comunicação, ordem de execução e sincronização. Consequentemente, sempre que nos deparamos com um defeito em software distribuído nos perguntamos: Por qual máquina começar? Qual componente originou o problema?

Uma vez claro que existe um defeito, é preciso identifica-lo e descobrir como resolvê-lo. A atividade de tentar descobrir as causas do defeito em um software já existente e tentar, de alguma maneira, consertá-lo, é conhecida como depuração. Embora a depuração seja considerada mais arte do que ciência [3], quando nos de-

paramos com um defeito em software, o modelo de raciocínio comumente adotado é: observar, coletar informações, elaborar uma hipótese, experimentar a hipótese e, se não houver sucesso, reiniciar o ciclo. Esse modelo de raciocínio é conhecido como *depuração cíclica*.

1.1 Ciclo de depuração de um software

A depuração cíclica conta com a condição de que devemos ser capazes de reproduzir o defeito quantas vezes forem necessárias. Somente a partir dessa condição é que podemos iniciar a procura pelo defeito [4]. Hailpern [1] descreve a depuração cíclica como uma composição de três atividades distintas: (i) depuração, (ii) verificação e (iii) testes.

i. **Depuração**: Atividade que envolve analisar, e possivelmente modificar, um programa que não atende as especificações. Seu propósito é localizar e corrigir o trecho de código responsável por violar uma especificação.
ii. **Verificação**: Atividade que procura provar ou demonstrar que o software satisfaz corretamente as especificações, capturando o modelo comportamental do programa, através de uma linguagem formal ou pelo uso do programa em si.
iii. **Testes**: Atividade que procura por casos onde o software não atende as especificações. Oposto à verificação, os testes tentam provar que o software não atende as especificações. Qualquer atividade que exponha o software a não atender as especificações é chamada de teste.

Normalmente, para depurar um software sequencial por intermédio de um depurador sequencial, paramos a execução do software a cada instrução, fazemos experimentos, corrigimos os possíveis problemas, e testamos novamente em seguida. Entretanto, em ambientes distribuídos, por ser executado em máquinas distintas, existe certa independência entre os processos. Por este motivo, devemos optar por um caminho que não prejudique o funcionamento dos componentes do software distribuído. Logo, precisamos de uma nova forma de pensar sobre como proceder à depuração para este caso.

1.2 Depuração aplicada a sistemas distribuídos

O principal desafio da depuração distribuída é não interferir no funcionamento do software distribuído. Interferências podem levar o software a não se comportar da mesma maneira de quando foi detectado o defeito. Porém, a única maneira de não interferir no funcionamento do software e reproduzir, da melhor forma pos-

sível, à condição que ocasionou o defeito seria monitorar sua execução, coletar informações, e inferir o ocorrido.

Bates [5] define que ferramentas de depuração clássicas não são apropriadas para depurar sistemas distribuídos. Tais ferramentas foram desenvolvidas usando o conceito de que todo software é executado em apenas uma máquina, e por este motivo seguem, essencialmente, as seguintes premissas:

- **São baseadas no estado corrente:** Acredita-se que erros são provocados por uma ou várias transições de estado erradas. Não são esperadas influências externas no estado.
- **As transições de estados dependem de instruções individuais:** O depurador tem que monitorar, a cada instrução, as variáveis e o fluxo de execução do software. Além disso, devido a instrumentação embutida no código, o depurador ainda é capaz inferir a próxima instrução e quais variáveis foram alteradas.
- **Há controle absoluto sobre o estado do software:** Para ser capaz de executar uma instrução por vez, monitorar variáveis e fluxos de execução, o depurador deve ser capaz de parar o software em qualquer ponto, além de ter acesso completo à memória utilizada pelo software em execução.

Algumas das propriedades dos sistemas distribuídos entram em conflito com as características listadas acima, logo, não são aplicáveis. Por exemplo, não existe como exercer controle absoluto sobre um software distribuído devido à autonomia existente entre os processos que o compõe. Qualquer tentativa de parar um processo individual implica em dessincronizar todo o software e mascarar o defeito que se deseja identificar.

Por essa razão, Bates propõe um mecanismo de depuração diferente para software distribuído. Segundo ele, a depuração é essencialmente baseada em abstrações, porém, em software distribuído, a quantidade de abstrações necessárias são demasiadamente complexas. Por isso, devemos criar ferramentas que nos auxiliem neste processo de abstração. Dessa forma, devemos observar o software como um gerador de fluxo de informações (mensagens de log, por exemplo) e representar por esses fluxos comportamentos significantes. Assim, com o objetivo de restringir as possíveis causas dos defeitos, temos condições de aplicar filtros à procura de comportamentos não adequados.

1.3 Objetivos

O objetivo deste trabalho é incrementar, de maneira não intrusiva à aplicação, as informações coletadas para a depuração de um sistema distribuído, auxiliando a identificação e facilitando a elaboração de um ambiente apropriado à reprodução dos defeitos encontrados. A maneira proposta por este trabalho alcançar tal

objetivo é capturar, com o auxílio do sistema operacional, mensagens trocadas pelos componentes do software em forma de pacotes, de forma similar a um analisador de protocolos de rede, e exibir tais pacotes usando a interface de um analisador de protocolos já existente. As técnicas de recomposição e decodificação de pacotes já existente no analisador serão exploradas pela ferramenta com o objetivo de investigar o conteúdo das mensagens trocadas pelos componentes, e assim, auxiliar o entendimento do defeito e sua depuração.

Dessa forma, a ferramenta proposta por este trabalho é separada em três camadas: módulo de carregamento dinâmico, interface do modo usuário com o núcleo e interface GUI com o usuário.

Para haver a monitoração transparente de processos, foi incorporado ao núcleo do sistema operacional, também conhecido como *kernel*, através de um módulo de carregamento dinâmico, um determinado código[1] capaz de monitorar a comunicação entre componentes. Com isso, os objetivos da monitoração são: Monitoração transparente das operações de comunicação dos componentes, representados por processos independentes do sistema operacional; Habilidade de selecionar em qual componente (equivalente a um processo) a monitoração deve ser habilitada, em tempo de execução, sem a necessidade de reiniciar o componente, nem alterar seu modo de inicialização; Exportar os dados monitorados para o usuário.

A existência de uma separação de privilégios entre o núcleo e o "modo usuário" (*user mode*) faz com que um programa em modo usuário não seja capaz de acessar diretamente a região de memória de uso exclusivo do núcleo. Assim, um dispositivo provido pelo núcleo se encarrega de realizar a comunicação entre ambos. Porém, os dados obtidos do núcleo contêm apenas dados brutos, logo, é necessário converter tais dados em um formato adequado à utilização pela interface GUI. Dessa forma, esta camada se encarrega da comunicação e da conversão dos dados obtidos. Assim, seus objetivos correspondem a: Obter do núcleo, usando o dispositivo provido por este, o resultado da monitoração; Converter os dados brutos fornecidos pelo núcleo em um formato intermediário capaz de ser interpretado pela interface GUI; Armazenar os dados convertidos no sistema de arquivos.

A interface GUI exibe de forma mais amigável o resultado da monitoração realizada pelo núcleo sobre os processos. Assim, com esta interface será possível: Visualizar a troca de mensagens armazenadas pela monitoração e as ordenando em função do momento em que o evento ocorreu; Filtrar mensagens com base nos componentes de origem, destino, horário e conteúdo; Inspecionar mensagens, visualizando seu conteúdo.

[1] Código representa instruções, em código fonte ou sua representação binária, para a realização de uma tarefa pelo computador.

1.4 Contribuições

A contribuição deste trabalho é uma ferramenta de monitoração para auxiliar a depuração em sistemas distribuídos, com baixa degradação de desempenho e com uma forma amigável de visualização da troca de mensagens entre componentes, ordenando a troca de mensagens em ordem causal e as exibindo na interface de visualização.

Este trabalho, diferente dos demais conhecidos nesta área, funciona no núcleo do sistema operacional, tornando desnecessário o carregamento de bibliotecas e a reinicialização dos processos os quais se deseja monitorar. Em contrapartida, é necessário ter acesso ao código fonte do núcleo do sistema operacional, conhecer o número de identificação do processo *(pid)*, e também ter acesso, com privilégios de administrador, às máquinas onde se deseja habilitar a monitoração.

1.5 Estrutura do documento

Este documento está estruturado da seguinte forma. No capítulo 2 serão mostradas as técnicas e as metodologias utilizadas para a depuração de software. No capítulo 3 mostraremos o desenvolvimento da ferramenta proposta por este trabalho. No capítulo 4 serão apresentadas as experimentações e os resultados obtidos, fechando com as conclusões no capítulo 5.