GOD: Um Depurador Simbólico para Sistemas de Objetos Distribuídos

Giuliano Mega e Fabio Kon

Departamento de Ciência da Computação – Universidade de São Paulo (USP)

{giuliano,kon}@ime.usp.br

http://eclipse.ime.usp.br/DistributedDebugging

Abstract. *Debugging distributed applications can be a difficult task. The lack of an observable global state and the intricate nature of distributed executions translate themselves into causal structures which are difficult to expose and analyse without proper assistance. The distributed thread concept, core to causality among synchronous-call, distributed systems, could be explored in building a tool for debugging the ever so popular distributed object systems. This paper presents a unique Java tool whose core consists of an extended symbolic debugger capable of tracking distributed threads on-line.*

Resumo. *Depurar aplicações distribuídas pode ser uma tarefa difícil. A não-existência de um estado global facilmente observável conjugada à natureza complexa das execuções distribuídas traduzem-se em estruturas causais que são difíceis de expor e analisar sem a assistência de ferramentas adequadas. O conceito de thread distribuído, que ocupa um importante papel na formação de vínculos causais em sistemas que utilizam chamadas remotas síncronas, poderia ser explorado na construção de um depurador voltado aos populares sistemas de objetos distribuídos. Este artigo apresenta uma ferramenta Java cujo núcleo consiste em um depurador simbólico estendido, capaz de rastrear threads distribuídos on-line.*

1. Introdução

Depuração é o processo através do qual são isoladas as causas de comportamentos errôneos em sistemas de software. Invariavelmente, depurar uma aplicação envolve fases de coleta e análise de informações em uma ou mais execuções. Embora a natureza dos dados e das formas de análise possa variar consideravelmente com a aplicação e com as ferramentas empregadas, o propósito dessas atividades é, via de regra, bem-definido: obter informações em quantidade suficiente para que seja possível reconstruir uma aproximação da execução da aplicação e, a partir daí, analisar essa reconstrução em busca de alguma violação de premissas capaz de produzir o comportamento errôneo observado.

Nos depuradores simbólicos convencionais como o GDB [1], por exemplo, informações como a posição dos *threads* da aplicação ou valores de variáveis de interesse são coletadas a todo instante e exibidas na tela para que o usuário (em geral o desenvolvedor da aplicação) possa analisá-las e tirar suas próprias conclusões. Ferramentas distintas podem gerar informações mais ou menos abstratas, de acordo, em

geral, com as abstrações que trabalham a linguagem ou ambiente para o qual foram projetadas. Algumas ferramentas, ainda, são capazes de conduzir formas limitadas de análise automática, em geral por meio da verificação de certos tipos de predicados e propriedades. Um exemplo disso é o mecanismo de detecção de *deadlocks* embutido na implementação da JVM da Sun [2].

Os sistemas distribuídos ocupam um lugar de destaque dentre os sistemas de software por apresentarem uma infinidade de opções para aplicação, cada qual portadora de características muito peculiares. O paradigma distribuído é conhecido por viabilizar, por exemplo, a construção de sistemas escaláveis, confiáveis, robustos e de alta disponibilidade. Uma outra categoria bastante relevante de aplicações distribuídas são aquelas destinadas a tirarem proveito da capacidade ociosa de grandes grupos estações de trabalho [3,4]. Infelizmente, apesar de todos os potenciais benefícios, esses sistemas são também conhecidos por serem notoriamente difíceis de construir.

Os primeiros sistemas distribuídos eram baseados em mecanismos bastante primitivos de passagem de mensagens. Justamente por serem pouco sofisticados, esses mecanismos deixavam a cargo do desenvolvedor muitos dos aspectos e idiossincrasias relativos à comunicação em ambientes heterogêneos. Seguindo a evolução natural dos paradigmas de projeto e codificação, foram desenvolvidos arcabouços (sob a forma de *middleware*) que isolam o desenvolvedor de parte dessa complexidade. Dentre os arcabouços para a construção de sistemas distribuídos mais bem-sucedidos e difundidos encontram-se os sistemas de *middleware* orientados a objetos (CORBA, Java/RMI e .NET, por exemplo). Sistemas construídos em cima desses arcabouços são conhecidos como *sistemas de objetos distribuídos*.

Infelizmente, depurar sistemas distribuídos não se tornou uma tarefa mais fácil por causa dos sistemas de *middleware*. Na verdade, é possível argumentar justamente em favor do contrário – esses arcabouços acoplam-se à aplicação do usuário por meio de código gerado automaticamente. Essa estratégia funciona muito bem durante as fases de desenvolvimento – o usuário do arcabouço pode acessar objetos locais e remotos de maneira uniforme enquanto a complexidade da infra-estrutura é elegantemente encapsulada por interfaces. Ao utilizar um depurador simbólico, no entanto, a coisa muda de figura. Toda aquela elegância e uniformidade desaparecem no instante em que se penetra no primeiro *stub*.

Este artigo descreve parte da arquitetura e da implementação de um depurador simbólico estendido para sistemas de objetos distribuídos. Um dos principais objetivos do nosso trabalho é produzir uma ferramenta que concilie depuração simbólica, sistemas distribuídos e *middleware*. O restante deste artigo está organizado da seguinte forma. A próxima seção descreve em mais detalhes parte da problemática ligada à depuração de sistemas distribuídos e motiva algumas das decisões de projeto da ferramenta. A Seção 3 apresenta uma visão geral da arquitetura da ferramenta e descreve os participantes mais relevantes. A Seção 4 sumariza nossos resultados e a Seção 5 conclui a exposição.

2. Motivação

Aplicações são depuradas para que a causa de um comportamento errôneo possa ser identificada. A atividade consiste, informalmente, em navegar através de uma

reconstrução aproximada da execução de uma aplicação até que, em algum instante, alguém (humano ou máquina) se depare com uma violação de alguma hipótese estabelecida. Reconstruir a execução de uma aplicação distribuída, no entanto, requer algum zelo – a ausência de relógios globais torna a identificação da ordem relativa dos eventos produzidos pelos vários nodos mais difícil. Essa identificação é crucial para que possamos reconstruir uma aproximação minimamente consistente da execução da aplicação distribuída [6]. Mais do que isso, gostaríamos de exibir essa aproximação a um usuário – o desenvolvedor da aplicação, por exemplo – para que ele possa analisá-la e tirar suas próprias conclusões, de maneira muito semelhante ao que acontece nos depuradores simbólicos tradicionais para aplicações centralizadas. Isso implica que a ferramenta deve prever em seu projeto uma localização central que, atuando como ponto de observação, permita que essa aproximação de execução seja facilmente acessada.

Um aspecto da análise das informações coletadas que estivemos cuidadosamente deixando de lado até então diz respeito ao momento em que essas informações devem ser analisadas. Existem dois extremos: de um lado situam-se as análises *post mortem*, em que os dados gerados pela execução da aplicação (traços) são armazenados e analisados somente após o término da execução da aplicação. No outro extremo situam-se as análises *on-line*, em que os dados produzidos são consumidos pelo mecanismo de análise tão logo estejam disponíveis. O nosso objetivo – aproximar a experiência de uso da ferramenta da experiência de uso de um depurador simbólico – implica que a única análise de fato obrigatória é a análise que correlaciona as informações recebidas e reconstrói a execução. Dessa análise resultam uma série de *cortes consistentes* [6], que devem ser exibidos *on-line* (i.e. conforme tornam-se disponíveis) ao desenvolvedor.

O efeito que desejamos obter consiste em dar ao usuário a ilusão de estar observando um sistema *multithreaded* – um objetivo muito próximo daquele do *middleware* orientado a objetos. Para esse propósito, optamos por expor os sucessivos estados do sistema distribuído sob a forma de *threads distribuídos*. *Threads distribuídos* são uma conseqüência natural do mecanismo de chamada síncrona e bloqueante empregada nos sistemas de *middleware* orientado a objetos e representam uma alternativa bastante conveniente para a visualização do estado de sistemas de objetos distribuídos. Informalmente, um *thread* distribuído é uma cadeia ordenada $(t_1,...,t_n)$ de *threads* locais (não necessariamente pertencentes ao mesmo processo) em que, para todo k, $1 \le k \le n$, t_k é um *thread* que faz uma chamada remota (cliente) e t_{k+1} é o *thread* que trata a chamada de t_k no objeto remoto (servidor). Parte do trabalho de criar a ilusão de que o sistema de objetos distribuído é na verdade um sistema *multithreaded* consiste em esconder do usuário os trechos de código relacionados às entranhas do *middleware*.

Gostaríamos, portanto, de uma ferramenta capaz de expor, *on-line*, uma série de cortes consistentes sob a forma familiar da execução dos depuradores simbólicos, escondendo do usuário as partes da execução relacionadas ao código do *middleware.*

3. Arquitetura da ferramenta

Conforme mencionamos anteriormente, é necessário prover um ponto central através do qual o usuário deve ser capaz de observar a execução do sistema distribuído. Esse

requisito induz uma arquitetura centralizada, bastante comum em ferramentas de depuração distribuída [7].

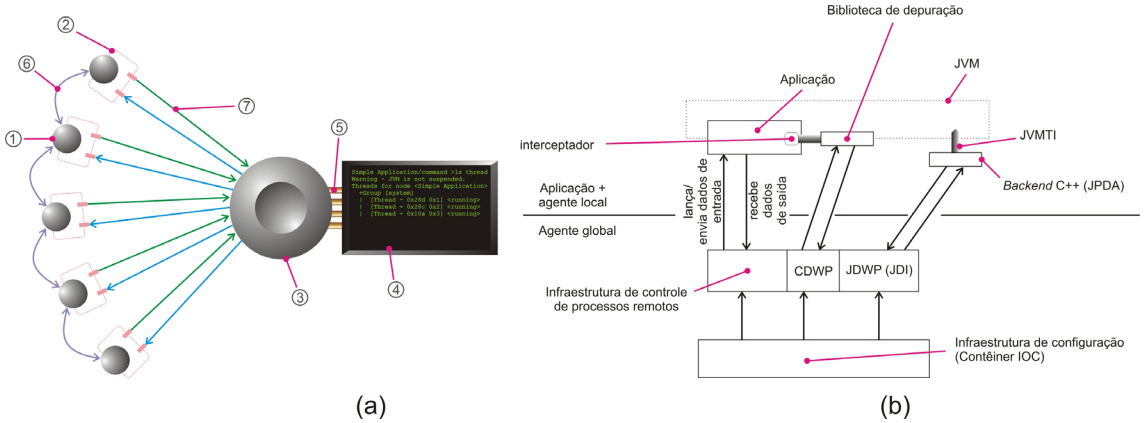


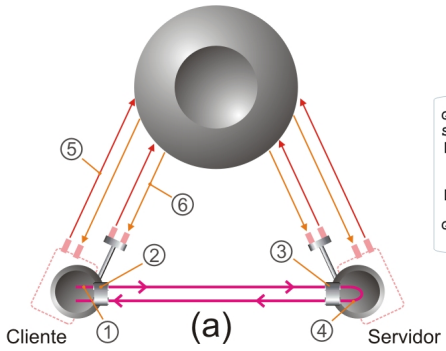
Figura 1. (a) visão geral da arquitetura da ferramenta e (b) visão mais detalhada

A Figura 1 (a) mostra uma visão geral da arquitetura da ferramenta. A cada processo componente do sistema distribuído **(1)** é atribuído um agente local **(2)**. Os agentes locais são responsáveis por penetrarem na execução de cada processo e atuarem como procuradores de um agente global **(3)**, coletando informações e interagindo com os processos da aplicação conforme o necessário. Os agentes locais produzem informações e respondem a certos comandos, definidos em protocolos de comunicação específicos da linguagem **(7)** em que é escrito o processo componente sob sua responsabilidade. Os protocolos de comunicação adotados pelos agentes locais podem variar, ficando a cargo do agente global lidar com eventuais diferenças. A ferramenta conta ainda com uma infraestrutura de gerenciamento remoto de processos, visível na Figura 1(b), cujo intuito é facilitar o processo de montagem de cenários de teste (possivelmente automatizados) e depuração. Essa infraestrutura é capaz de controlar o ciclo de vida de processos remotos (lançando e finalizando processos) e de comunicar-se com a entrada e saída padrão desses processos, reforçando a conveniência de uma visão centralizada do sistema distribuído. Em alguns sistemas é possível, ainda, comunicar-se com as interfaces gráficas de processos remotos.

Os agentes locais são, de fato, mais complexos do que aparentam. Isso porque, para grande parte das linguagens, já existem depuradores simbólicos prontos (por exemplo o GDB e a *Java Platform Debug Architecture* ou *JPDA* [8]) capazes de desempenhar grande parte das funções de coleta e intervenção de execução requeridas pela nossa ferramenta (i.e. *breakpoints*, execução passo-a-passo, inspeção de variáveis e *threads* locais). Esses depuradores, no entanto, são míopes com relação ao *middleware* e a diversos aspectos da distribuição do sistema que nos são de interesse. É preciso, portanto, capturar de alguma forma as informações extras necessárias. Modificar o código desses depuradores nem sempre é possível (a implementação da plataforma de depuração Java (JPDA) da Sun, por exemplo, é de código fechado) ou desejável (por questões de portabilidade entre JVMs, por exemplo). Acabamos, portanto, por optar por uma abordagem baseada em instrumentação: costuramos ao código da aplicação uma coleção de interceptadores, responsáveis pela extração de informações relevantes da execução nos momentos apropriados e pelo envio, por meio de uma biblioteca, desses dados ao agente central. A implicação imediata disso é que os agentes locais

comunicam-se, a princípio, por meio de dois protocolos distintos: um deles específico do arcabouço de depuração remota provido pela linguagem (ou ferramenta de depuração remota construída para a linguagem) e outro específico do depurador distribuído. A Figura 1 (b) mostra a versão atual da ferramenta, com seus dois protocolos – o CDWP, protocolo específico do depurador distribuído (batizamo-no de *Custom Debug Wire Protocol* ou *CDWP*) e o JDWP (*Java Debug Wire Protocol*), específico da JPDA. Espera-se que a adição de novas linguagens corresponda à adição de novos protocolos ao agente global – essa abordagem parece, pela nossa experiência, mais simples do que adaptar os agentes locais para que conversem todos o mesmo protocolo.

O agente global, por sua vez, é responsável por 1) controlar os agentes locais quando necessário, 2) correlacionar as informações provenientes dos diversos agentes locais e 3) construir uma aproximação causalmente consistente da execução do sistema. A visão aproximada da execução é disponibilizada por meio de uma API **(5 na Figura 1(a)).** A versão atual do sistema conta com uma interface simples de modo texto (no estilo GDB) que interage com essa API **(4 na Figura 1(a))**. Existem diversos problemas associados à construção de um depurador interativo que trabalha com aproximações de execuções. Um dos mais fundamentais diz respeito a como conciliar comandos interativos com uma representação que está sempre em atraso com o estado "real" da aplicação. A ferramenta aborda o problema conduzindo uma série de sincronizações estratégicas (que possuem, por sua vez, seus próprios efeitos colaterais). Esse problema, no entanto, está fora do escopo deste artigo e fica para artigos futuros. A ferramenta permite rastrear o estado do sistema distribuído on-line. Mais do que isso, é possível interagir com os *threads distribuídos* num estilo muito semelhante ao permitido pelos depuradores simbólicos convencionais em sistemas centralizados. Isso inclui suspender e retomar a execução dos *threads distribuídos* e inspecionar as suas pilhas de execução, com toda a informação simbólica disponível nelas.




```
Global/command >show stack 0.2
Stack information for distributed thread [0.2]
[Servidor CORBA 1]<---+---->0 Stock._QuoterStub.value(java.lang.String)
                      +---->1 implementations.QuoterImpl.value(java.lang.String)
                      +---->2 Stock._QuoterStub.value(java.lang.String)
[Cliente CORBA 1]<----|     3 Client$RunnableRequester.run()
                      +---->4 java.lang.Thread.run()
Global/command>
```

(b)

Figura 2. (a) conjunto em ação e (b) a interface modo texto

A Figura 2 (a) mostra o conjunto em ação num cenário simples – um cliente **(1)** faz uma chamada a um método de um objeto remoto (servidor), localizado em outro processo **(4).** Após alguns passos no modo de execução passo-a-passo, a listagem da pilha virtual do *thread* distribuído, originado no cliente, é mostrada na Figura 2 (b). O protocolo específico de linguagem/plataforma de depuração **(5)** é utilizado para a interação e controle de execução remotos dos processos que constituem a aplicação distribuída – i.e. a execução passo-a-passo no nosso exemplo. O protocolo do depurador distribuído **(6)** é utilizado para a comunicação de eventos relativos ao estado do sistema distribuído não cobertos pela plataforma de depuração – é ele quem viabiliza a montagem pilha virtual. Informações adicionais a respeito de algumas das idéias por trás do mecanismo podem ser encontradas em [9].

4. Trabalhos em andamento e futuros

Este artigo apresentou parte do núcleo de uma ferramenta de depuração para sistemas de objetos distribuídos. Há ainda muito trabalho por fazer, em especial nas partes de análise de desempenho, adaptação para outras linguagens/*middleware* (a implementação atual só permite que se trabalhe com Java/CORBA) e interface com o usuário. Os trabalhos em andamento incluem a criação de uma interface de usuário amigável para a plataforma Eclipse [10] e a implementação de um arcabouço para a montagem de testes distribuídos automatizados, baseado na infraestrutura já presente. Trabalhos futuros incluem a implementação de facilidades para a re-execução controlada de aplicações distribuídas e para a detecção automática de predicados globais.

5. Conclusão

Depurar sistemas distribuídos é uma tarefa complexa. Nosso trabalho representa um passo à frente numa direção que acreditamos ser frutífera: tornar possível ao programador comum, por meio de uma noção intuitiva, a compreensão da complexa dinâmica das execuções distribuídas. A versão atual da ferramenta pode ser obtida livremente em http://eclipse.ime.usp.br/DistributedDebugging.

Referências

[1] GNU foundation. "The GNU project debugger." Internet: www.gnu.org/software/gdb/gdb.html, [May 20, 2005].

[2] Sun Microsystems. "Virtual Machine Enhancements in J2SE 1.4.x." Internet: http://java.sun.com/j2se/1.5.0/docs/guide/vm/enhancements.html

[3] SETI @ home website. http://setiathome.ssl.berkeley.edu [May 19, 2005].

[4] A. Goldschleger, F. Kon , A. Goldman, M. Finger and G. C. Bezerra. (2004, Nov.) "InteGrade: object-oriented middleware leveraging idle computing power of desktop machines." *Software: Practice & Experience*. vol. 16, pp. 449-459. March, 2004.

[5] L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21(7):558-565, July 1978.

[6] R. Schwartz and F. Mattern. "Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail". *Distributed Computing*. vol. 7, pp. 149-174, July 1994.

[7] IBM Corporation. "IBM Distributed Debugger: Overview." Internet: http://aixdoc.urz.uniheidelberg.de/doc_link/en_US/local/XLFortran.7.1/html/debugger/concepts/cbcdbovr.htm, [May 22, 2005]

[8] Sun Microsystems. "The Java Platform Debug Architecture." Internet: http://java.sun.com/products/jpda/index.jsp. [May 22, 2005]

[9] G. Mega and F. Kon. "Debugging Distributed Object Applications with the Eclipse Platform," in *Proc. of the 2004 ACM OOPSLA Eclipse Technology Exchange Workshop*, 2004, pp 47-51.

[10] Eclipse website. http://www.eclipse.org. [May 22, 2005].