Tolerância a Falhas em Sistemas de Agentes Móveis

Tiago Fioreze, Ingrid Jansch-Pôrto, Lisandro Zambenedetti Granville

Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – 90501-970 Porto Alegre, RS

{tfioreze, ingrid, granville}@inf.ufrgs.br

Abstract. *Mobile agents belong to an important area of distributed processing. Fault tolerance techniques are required for robust and reliable applications based on mobile agents. The main purpose of this paper is to present fault tolerant approaches used with mobile agents. We start with the basic operations of the mobile agents, followed by some of the main problems faced by applications based on mobile agents. Then, some of the main techniques of fault tolerance in applications based on mobile agents are explained and discussed.*

Resumo. *Agentes móveis fazem parte de uma importante área no campo de processamento distribuído. Técnicas de tolerância a falhas são fundamentais para aplicações robustas e confiáveis baseadas em agentes móveis. O objetivo principal deste artigo é apresentar métodos de tolerância a falhas usados em sistemas de agentes móveis. Este artigo apresenta inicialmente o funcionamento básico dos agentes móveis, seguido de alguns dos principais problemas enfrentados por aplicações baseadas em agentes móveis. Posteriormente, algumas das principais técnicas de tolerância a falhas em aplicações baseadas em agentes móveis serão explicadas e discutidas.*

1. Introdução

Agentes móveis são programas que não estão ligados exclusivamente ao sistema que tenha iniciado sua execução. Eles têm a liberdade de viajar entre computadores pertencentes a uma rede. Criados em um ambiente de execução, eles podem transportar seus estados e códigos para qualquer outro ambiente de execução, onde eles podem voltar a executar [Lange and Oshima, 1998]. Falhas nesses ambientes de execução podem levar para uma perda parcial ou, até mesmo, completa de um agente. Como agentes móveis podem executar em pequenas e grandes redes (e.g. Internet), é plausível que eles enfrentem problemas relativos a atrasos, falhas de comunicação e demais problemas existentes em redes de computadores. Devido a isso, é difícil ao proprietário do agente saber quando um agente criado foi perdido ou se a execução do agente está atrasada devido a problemas no meio de comunicação. Essas incertezas podem originar as seguintes situações:

- o proprietário do agente pode acreditar que o agente criado foi perdido, quando na verdade ele não foi. O proprietário, acreditando nisso, poderia recriar o agente perdido e lançá-lo novamente na rede, o que poderia implicar em execuções simultâneas do mesmo agente em determinados dispositivos.

- O proprietário do agente fica aguardando pelo término da execução, mas o agente foi perdido. Isso caracteriza uma situação bloqueante, uma vez que o proprietário fica bloqueado aguardando a execução do seu agente.

Tolerância a falhas em sistemas de agentes móveis visa eliminar, ou pelo menos minimizar, essas incertezas, seja através da garantia de que o agente chegue ao seu destino ou, no mínimo, através da notificação ao proprietário do agente de problemas em potencial [Pleisch and Schiper, 2000]. Existem várias técnicas de tolerância a falhas aplicadas em sistemas de agentes móveis. Neste artigo, serão apresentadas algumas das técnicas mais conhecidas e relatadas no meio científico. Este artigo não visa indicar qual técnica é a melhor, mas sim expor as características e o funcionamento das técnicas estudadas. As técnicas abordadas neste artigo são: as técnicas baseada em *checkpointing*, em replicação e em replicação com votação.

O restante deste artigo está organizado da seguinte forma. A seção 2 apresentará o funcionamento dos agentes móveis e quais são os tipos de falhas que podem acometê-los. A seção 3 mostrará algumas das técnicas de tolerância a falhas em sistemas de agentes móveis mais conhecidas. Finalmente, a seção 4 conclui este artigo.

2. Agentes Móveis

Basicamente, agentes móveis são programas autônomos de computador que viajam através de um rede de computadores heterogêneos. A figura 1 ilustra diferentes estágios (k) da execução de um agente móvel (a_i) em uma seqüência de máquinas (p_j).

Figura 1: Funcionamento básico de um agente móvel.

Para um melhor entendimento referente ao funcionamento dos agentes móveis, alguns conceitos são importantes.

- **Agente:** Um processo autônomo que viaja entre diferentes computadores de uma rede realizando procedimentos para o qual foi instruído e que é capaz de migrar seu código de uma máquina a outra de forma independente.
- **Lugar:** Refere-se ao ambiente de execução para um agente móvel arbitrário. Este ambiente deve fornecer recursos necessários à execução do agente móvel.
- **Estágio:** Refere-se à fase da execução de um agente móvel arbitrário, em uma localização específica.

Infelizmente não existe *software* ou *hardware* que seja totalmente imune a falhas. Com isso, qualquer ambiente de execução é potencialmente sujeito a falhas. Existem três tipos de defeitos que podem afetar um agente. São eles:

- **Colapso do próprio agente:** É a perda completa do agente móvel. Esse tipo de defeito geralmente não causa o colapso do ambiente de execução ou do computador no qual o agente móvel está realizando a execução de tarefas.

- **Colapso no ambiente de execução:** Um agente móvel precisa de um ambiente propício a sua execução. Se tal ambiente de execução entrar em colapso, por conseqüência o agente móvel sofrerá o mesmo problema (perda total).
- **Colapso no computador:** Caracteriza-se pela incapacidade do computador se comunicar com os demais computadores de uma rede e também de realizar processamento de tarefas. O colapso no computador leva, por conseqüência, ao colapso do ambiente de execução e do agente móvel.

Salienta-se que, embora dificilmente o colapso repercuta diretamente nos níveis superiores, uma execução mal-sucedida de um agente pode levar o ambiente de execução ou o computador no qual ele estava executando para um estado inconsistente. Logo, necessita-se o uso de mecanismos que desfaçam ações impróprias realizadas por agentes, de modo que o computador ou o ambiente de execução volte a um estado consistente.

3. Técnicas de Tolerância a Falhas em Sistemas de Agentes Móveis

Agentes móveis estão sendo explorados em várias áreas, tais como: comércio eletrônico, gerenciamento de redes e computação distribuída. Apesar disso, eles serão somente usados em larga escala se alguns importantes problemas (tolerância a falhas e segurança) puderem ser resolvidos de um modo eficaz [Silva et al., 2000]. O que será visto nesta seção são algumas técnicas de tolerância a falhas em sistemas de agentes móveis.

3.1. Técnica baseada em *Checkpointing*

Uma das técnicas usadas na tolerância a falhas em sistemas de agentes móveis é baseada em *checkpointing*. Existem diferentes esquemas de *checkpointing* em agentes móveis [Yeom et al., 2002]. Este artigo não levará em conta as particularidades de cada um, mas sim a idéia básica do *checkpointing* em agentes móveis que é o armazenamento estável dos códigos e estados de um agente móvel arbitrário em um determinado computador.

A técnica baseada em *checkpointing* pode ser explicada informalmente como segue. Quando um agente móvel migra de um computador a outro, levando consigo seu estado atual e seu código, o computador destino armazena essas informações em um local seguro (e.g. sistema de disco) antes de iniciar a execução do agente. Com isso, o agente não é perdido caso o computador destino falhe, já que existem informações armazenadas no computador que permitem ao agente ser executado novamente após a recuperação do computador destino.

Entretanto, enquanto o computador destino estiver desligado, o procedimento de execução do agente ficará bloqueado, e por conseqüência, o processo no computador que criou o agente (fonte do agente) ficará bloqueado esperando o fim da execução da agente, ou seja, esta técnica de *checkpointing* é bloqueante.

3.2. Técnica baseada em Replicação

Diferentemente da técnica vista na seção 3.1, a técnica baseada em replicação de agentes [Pleisch and Schiper, 2000] evita que a execução de um agente móvel seja bloqueada através da criação de várias cópias do mesmo agente em um determinado estágio. Ao invés de enviar um agente para um único lugar em um determinado estágio, o agente é copiado para um grupo M_i de lugares $\{p_i^0, p_i^1, p_i^2, ...\}$. Para prevenir que o colapso de uma

máquina afete múltiplos lugares em um estágio S_i, cada lugar p_i^j (j=0,1,...) é normalmente localizado em computadores diferentes. Se um determinado lugar falhar, a execução do agente em um estágio S_i continua em outro lugar. Por exemplo, a figura 2 mostra um defeito no lugar p_2^0, mas a execução do agente continua, uma vez que os lugares p_2^1 e p_2^2 estão aptos a executarem o agente.

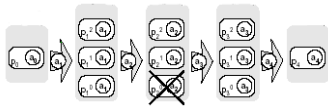

Figura 2: Exemplo de falha no funcionamento de um agente móvel.

Embora esta técnica resolva o problema de bloqueio na execução de um agente móvel, ela é propensa a violar a propriedade dos agentes móveis executarem somente uma vez (*exactly-once property*) em determinado estágio. Uma falsa suposição que o lugar p_2^0 tenha falhado pode levar a execução do agente a_2 no lugar p_2^1, por exemplo, enquanto o mesmo agente continua sendo executado em p_2^0. Isso leva para múltiplas execuções de operações de um agente móvel.

3.3. Técnica baseada em Replicação com Votação

No método de replicação baseado em votação [Rothermel and Strasser, 1998], as ações de envio de um agente para determinado lugar, da execução desse agente nesse lugar e do recebimento de um agente em um próximo lugar fazem parte de uma única transação.

Similar ao método de replicação, a técnica de replicação baseada em votação mantém um grupo M_i de lugares $\{p_i^0, p_i^1, p_i^2, ...\}$ associados a um estágio S_i. A diferença é que, nessa técnica, cada lugar tem uma propriedade associada a ele, o que define uma ordem entre os lugares pertencentes ao mesmo estágio. Cada estágio inicialmente seleciona o lugar com a maior prioridade sendo este denominado de **trabalhador**. Os demais lugares são denominados de **observadores** e têm como função monitorar a disponibilidade do trabalhador. Quando os observadores notam que o trabalhador entrou em colapso, eles selecionam um novo trabalhador entre eles mesmos, através de um protocolo de eleição.

Para preservar a propriedade dos agentes móveis executarem somente uma vez em determinado estágio, um processo de votação foi integrado a um protocolo de *commit* de duas fases [Gray and Reuter, 1994]. Com essa integração, um trabalhador pode somente finalizar uma transação se a grande maioria dos lugares envolvidos na execução de um agente móvel concordarem.

A arquitetura projetada por Rothermel e Straber (1998) consiste de um **gerenciador de transação** que executa o protocolo 2PC (*commit* de 2 fases) e **gerenciadores de recursos** que mantêm dados recuperáveis. Nessa arquitetura, o gerenciador de transação do trabalhador interage com um outro tipo de gerenciador de recursos, denominado **orquestrador**. O orquestrador, que se comunica com os **eleitores** pertencentes ao seu estágio de execução, é responsável pela organização da votação.

3.3.1. Protocolo de Votação

O protocolo de votação é executado entre o orquestrador e o eleitor de um estágio S_i. Primeiramente, o orquestrador envia uma requisição chamada de `VOTE` (voto) para cada eleitor do seu estágio de execução. O formato da requisição é o seguinte: `VOTE(StageId, Tid, OrchId)`. Nessa requisição estão incluídos o ID (identificador) do estágio atualmente sendo processado (`StageId`), o ID do orquestrador do processo de votação (`OrchId`) e o ID da transação que o orquestrador está manipulando (`Tid`). Ao final do envio das requisições aos eleitores, o orquestrador espera por respostas dos eleitores.

Cada eleitor possui um *buffer* denominado `OrchSet`, utilizado para armazenar o ID dos orquestradores que fizerem a requisição `VOTE`. Um eleitor determina seu voto ao orquestrador baseado no conteúdo do `OrchSet`. As seguintes situações são possíveis:

1. Se `OrchSet` estiver vazio, significa que o eleitor não votou ainda. Nesse caso, o ID do orquestrador é armazenado no `OrchSet` e uma resposta `YES(StageId, Tid, VoterId)` é enviada de volta ao orquestrador. O parâmetro `VoterId` identifica o eleitor votante.
2. Se `OrchSet` não estiver vazio, significa que existe mais de um orquestrador competindo pelo voto. Assume-se que *L* seja o lugar com maior prioridade em `OrchSet` . Se `OrchSet` não estiver vazio e `OrchId` possuir prioridade inferior a *L*, então o eleitor já votou `YES` para um lugar com maior prioridade. Nesse caso, o eleitor retorna `NO(StageId, Tid, VoterId)` ao orquestrador.
3. Se `OrchSet` não estiver vazio e o `OrchSet` possuir prioridade maior que *L*, significa que o eleitor já votou para um lugar com uma prioridade menor. O eleitor então envia ao orquestrador um sim 'condicional' denominado `COND_YES (StageId, Tid, OrchSet, VoterId)` e adiciona `OrchId` em `OrchSet`. Esse voto significa que o eleitor vota `YES`, contanto que todos os lugares armazenados em `OrchSet` também votem `YES`.

Depois de analisar todos os `COND_YES` e `YES` dos eleitores e verificar que eles constituem a maioria dos votos, o orquestrador retorna a mensagem *rm_yes* para o gerenciador de transações local informando que a votação transcorreu normalmente.

Na fase 2, se o gerenciador de transações finalizar a transação, ele envia a mensagem *rm_commit* para cada gerenciador de recursos local. Se o gerenciador de transações abortar a transação, ele envia a mensagem *rm_abort* para o orquestrador. Esse envia uma requisição denominada `UN_VOTE` para todos os eleitores que participaram da votação organizada por ele e que votaram `YES` ou `COND_YES`. Após isso, a transação abortada é reiniciada. Os eleitores, ao receberem a requisição `UN_VOTE`, removem o `OrchId` em `OrchSet`, permitindo que um lugar de menor prioridade seja prioritário caso algum lugar com maior prioridade falhe.

3.3.2. Protocolo de Detecção de Defeitos

O trabalhador de um estágio periodicamente envia mensagens `I_AM_ALIVE` para os observadores. Se um observador não receber as mensagens `I_AM_ALIVE` por um período de tempo pré-determinado, ele supõe que o trabalhador falhou e propõe uma eleição entre os

demais observadores para selecionar o novo trabalhador. Um observador envia uma mensagem ARE_YOU_THERE para todos os observadores com a prioridade maior. Os observadores que estiverem ativos responderão com uma mensagem I_AM_THERE. Se nenhuma mensagem chegar dentro de um tempo razoável, o observador que iniciou a eleição será o novo trabalhador. O novo trabalhador envia então a mensagem I_AM_SELECTED para os demais e a partir daí a execução de um agente móvel prossegue. Os demais participantes da eleição, ao receberem a mensagem I_AM_SELECTED realizam o trabalho de monitoramento do novo trabalhador.

Com esses protocolos, o método de replicação baseada em votação consegue evitar que uma execução de um agente móvel fique bloqueado, além de preservar a propriedade dos agentes móveis executarem somente uma vez em determinado estágio através do protocolo de votação.

4. Conclusão

Neste artigo foi apresentado o funcionamento básico dos agentes móveis e como falhas que ocorrem durante a execução dos mesmos podem influenciar de forma negativa um sistema baseado em agentes móveis. Foram apresentadas três técnicas de tolerância a falhas: as técnicas baseadas em *checkpointing*, em replicação e em replicação com votação. A primeira previne a perda de um agente, mas é dita bloqueante. A segunda evita o bloqueio da execução de um agente, mas é propensa a violar a propriedade de execução única em determinado estágio. A última técnica é um método robusto que evita o bloqueio da execução e preserva a propriedade dos agentes móveis executarem somente uma vez em determinado estágio. Finalmente, cabe lembrar que a vinculação a outros serviços tais como os de detecção de defeitos é extremamente importante na solução adequada das técnicas aqui apresentadas.

Referências

Gray, J. and Reuter, A. (1994). *Transaction Processing - Concepts and Techniques*. Morgan Kaufmann.

Lange, D. B. and Oshima, M. (1998). *Programming and Deploying Java Mobile Agents with Aglets*. Adison-Wesley.

Pleisch, S. and Schiper, A. (2000). Modeling fault-tolerant mobile agent execution as a sequence of agreement problems. In *Proceedings of the* 19^{th} *IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 11–20.

Rothermel, K. and Strasser, M. (1998). A fault-tolerant protocol for providing the exactly-once property of mobile agents. In *Proceedings of the* 17^{th} *IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 100–108.

Silva, L. M., Batista, V., and Silva, J. G. (2000). Fault-tolerant execution of mobile agents. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 135–143.

Yeom, H. Y., Kim, H., Park, T., and H.Park (2002). The cost of checkpointing,logging and recovery for the mobile agent systems. In *Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)*, pages 45–48.