Antonio Ribeiro Alves Júnior

Arquitetura de um Middleware para Simulação Distribuída

Itajubá - MG

01 de Junho de 2012

Antonio Ribeiro Alves Júnior

Arquitetura de um Middleware para Simulação Distribuída

Dissertação submetida ao Programa de PósGraduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação.

Orientador:
Prof. Dr. Edmilson Marmo Moreira

Universidade Federal de Itajubá - Unifei
Instituto de Engenharia de Sistemas e Tecnologias da Informação
Engenharia da Computação

Itajubá - MG

01 de Junho de 2012

Sumário

Lista de Figuras

1 Introdução

1.1 Simulação

A simulação é uma técnica que permite prever e visualizar o comportamento de sistemas reais a partir de modelos matemáticos. As aplicações da simulação abrangem diversos benefícios, tais como: a possibilidade de antever possíveis problemas ou comportamentos indesejáveis de um sistema, auxílio na tomada de decisão sem a necessidade de intervir no sistema real, facilidade na manipulação e alteração dos modelos, economia de recursos (físicos e financeiros) durante a tomada de decisões, dentre outros.

Para utilizar a simulação é necessário construir e analisar modelos que represente o sistema. Os modelos podem ser classificados de diferentes formas. Uma classificação pode ser considerada verificando a influência ou não de variáveis aleatórias no sistema. Os sistemas são ser representados por um modelo determinístico, quando estes podem ser considerado totalmente livre de aleatoriedade, ou estocásticos, quando estes consideram aleatoriedade.

Os modelos que descrevem o comportamento através do tempo podem ser classificados como contínuos e discretos no tempo. Nos modelos de estados contínuos, as variáveis de estados variam espontaneamente. Já nos modelos de estados discretos, as mudanças ocorrem em pontos específicos e descontínuos do tempo.

Este trabalho enfoca os modelos estocásticos e de estados discretos, uma vez que eles são os que melhor representam modelos de sistemas computacionais.

Figura 1.1: Simulação Sequencial.

Um sistemas de simulação sequencial, onde uma única máquina executa toda a simulação, pode ser retratado como uma fila de eventos aguardando para serem tratados. Cada evento possui o seu tempo de execução, como pode ser visto na figura 1.1, que deve ser obedecido para garantir consistência do resultado.

Neste modelo sequencial, o sistema responsável pela simulação retira o próximo evento da fila de execução para tratá-lo. Ao fim do processamento, um próximo evento é retirado da fila, e isto se repete até o final de lista de eventos futuros. O tratamento de um evento pode ou não resultar dados que sejam necessários em um processamento futuro.

1.2 Simulação Distribuída

A simulação é um processo que apresenta um custo computacional muito alto, tendo em vista a grande quantidade de dados que devem ser processados e a complexidade dos modelos matemáticos empregados. Esses fatores em conjunto podem encarecer computacionalmente o sistema, levando à ineficiência da simulação.

Uma das formas encontradas para se solucionar estes problemas foi dividir o tratamento dos diversos eventos entre vários processadores de uma mesma máquina paralela ou sobre um sistema distribuído, dando origem assim à Simulação Distribuída.

Distribuindo os eventos, reduz-se o tempo gasto pelos programas de simulação, mas, em contrapartida, novas situações necessitam de observação devido às características deste tipo de aplicação. É preciso sanar os problemas com a sincronização dos processos, sobrecarga da rede de comunicação, necessidade de balanceamento de carga do sistema, dentre outros.

Em um sistema de Simulação Distribuída, três estruturas devem ser observadas

no desenvolvimento da simulação orientada à eventos:

- As variáveis que descrevem os estados do sistema;
- Uma lista de eventos futuros, que contém os eventos a serem executados;
- Um relógio Global, que controla o progresso da simulação.

Os eventos devem ser executados obedecendo o seu *timestamp*. O programa de simulação deve remover repetidamente da fila o evento com a menor marca de tempo e executá-lo. Assim que um evento é retirado da fila de execução, o relógio global avança para o tempo de ocorrência do evento. Esse mecanismo garante que todos os eventos sejam executados obedecendo a ordem cronológica do tempo de simulação. Porém, em se tratando de um sistema distribuído, não há como haver uma fila única de eventos. Portanto, o sistema passa a ser dividido em n processos denominados $p_1, p_2, \ldots, p_n$, cada um representando um processo do sistema real. Novos mecanismos devem ser incorporados ao sistema de simulação para garantir que cada evento seja executado na sua devida ordem.

Para cada processo lógico é atribuído um relógio que indica o seu progresso na simulação. A comunicação entre os processos se dá através mensagens, uma vez que não há áreas de memória compartilhadas entre os processos. Estas mensagens são também responsáveis pela sincronização do sistema. Caso algum evento e_b venha a ocorrer antes de um segundo evento e_a, e sendo $a < b$, tem-se assim um erro de causa e efeito. Como em um sistema real nunca existirá tal situação, isto caracteriza uma inconsistência na simulação.

Os conceitos de sincronização de processos levaram ao desenvolvimento de protocolos, classificados como conservativos ou otimistas, para garantir a sincronização dos processos da simulação distribuída, evitando ou corrigindo erros de causa e efeito (**??**).

1.3 Objetivos

Este trabalho tem como objetivo avaliar a viabilidade de se implementar um protocolo de sincronização de simulação distribuída utilizando agentes móveis. Para isto é empregada a utilização da biblioteca de agentes móveis *Aglets* para a implementação do protocolo *Rollback* Solidário.

1.4 Organização da Monografia

Os capítulos seguintes abordam uma leve explicação do funcionamento dos protocolos de sincronização de simulação distribuída e de como o trabalho foi abordado para a sua implementação.

O capítulo dois trata dos protocolos de sincronização de eventos em simulação distribuída. Ele inicia com a explicação das principais diferenças entre protocolos conservativos e otimistas, e em seguida traz o princípio básico de funcionamento dos protocolos *Time Warp* e *Rollback* Solidário.

No terceiro capítulo são apresentados os agentes móveis e a biblioteca *Aglets*, assim como o seu funcionamento no contexto computacional.

O quarto capítulo trata a implementação do protocolo *Rollback* Solidário sobre a biblioteca *Aglets*, assim como o seu funcionamento.

Por fim, o capítulo número cinco discute as conclusões obtidas e as possibilidades de continuação deste trabalho.

2 Conclusões

A implementação do protocolo *Rollback* Solidário utilizando a biblioteca de agentes móveis *Aglets*, proposto nesse trabalho, trouxe algumas ferramentas para a inserção de um código de simulação distribuída sobre uma rede de computadores (sistema distribuído). A utilização de uma biblioteca de agentes móveis em Java foi útil na simplificação na comunicação e na migração dos processos.

Durante a execução deste trabalho, várias etapas individuais foram cumpridas. Em um primeiro momento, a preocupação foi a de se compreender de forma satisfatória o funcionamento de um sistema de simulação distribuído (simulação discreta, no contexto desse trabalho). Foram introduzidos os conceitos de simulação orientada a eventos, relógios lógicos, relógios vetoriais, *checkpoints*, precedência causal, mensagens e anti-mensagens, mensagens *straglers*, cortes consistentes, linhas de recuperação e, finalmente, o funcionamento do protocolo *Rollback* Solidário na sua forma semi-síncrona, com processo observador.

Um segundo passo foi o aprendizado da biblioteca *Aglets*. Nessa etapa foram desenvolvidas pequenas aplicações utilizando-se a biblioteca *Aglets* e o servidor de agentes *Tahiti*. O processo de aprendizado foi baseado no manual oficial da biblioteca(**??**). Foram desenvolvidas as técnicas de criação de *aglets* através do servidor, criação de *aglets* através de outros agentes já existentes, clonagem de agentes, migração, compreendimentos sobre o ciclo de vida de um *aglet*, envio, recebimento e tratamento de mensagens, dentre outros pontos.

2.1 Sobre a Implementação

No desenvolvimento do protocolo, um primeiro obstáculo encontrado foi a impossibilidade de se invocar métodos de outros agentes. Um agente se comunica com outro exclusivamente através de mensagens, mesmo estando em um mesmo contexto. Para se solucionar este problema, foi utilizada uma mensagem de disparo de método. Junto à essa mensagem, é enviada o parâmetro a ser tratado pelo método invocado, através do argumento arg da mensagem enviada. Com isso foi possível a invocação de métodos de outros agentes, estejam eles no mesmo contexto ou até mesmo em outro contexto.

Decorrente dessa solução de se enviar o parâmetro do método como argumento de uma mensagem, ocorreu a necessidade de se enviar por vezes objetos que nào suportavam serialização. Esses objetos não podem ser enviados como argumento de uma mensagem (é o caso do objeto *AgletID*). Para isso foi utilizado o método *toString()* em conjunto com o construtor *AgletID(String str)*, da classe *AgletID* . Com isso, o objeto *AgletID* foi, no agente de origem, convertido em uma *String*, enviado ao destino como *String* e, ao chegar no seu agente destino, foi recriado a partir desse argumento enviado como *String*.

Outros objetos, como o *AgletContext*, não suportam serialização. Isso serve, de certa forma, para que em uma migração de contexto, o abjeto *AgletContext* criado não venha a ser errôneamente utilizada. Uma forma contornar essa situação é a de apagar objeto *AgletContext* antes de uma migração e atulizá-lo assim que o objeto chega ao seu novo contexto.

Por fim, alguns métodos utilizados no manual estão marcados como *Deprecated*. Um desses métodos foi largamente utilizado nesse trabalho, o método *getAgletProxy(URL,AgletID)* da classe *AgletContext*. Esse método consiste em receber o ID de um agente e a URL do seu contexto e, com isso, gerar um *proxy* que possibilite a comunicação entre esses agentes. O problema criado por esse método é que, quando se é requisitado um proxy de um agente que não pertence a mesma URL,

durante a comunicação podem ocorrer falhas nos envios das mensagens. Para a solução desse problema foram testadas algumas alternativas:

- **Utilizar um método que realize a mesma tarefa:** Não foi encontrado em toda a documentação um método que possa substituir este método em questão, sendo essa alternativa descartada;
- **Serializar e enviar o proxy de comunicação:** O objeto *AgletProxy* não suporta serialização, o que impossibilitou essa solução;
- **Centralizar todo o tráfego para o processo observador:** Essa solução consiste em realizar a comunicação sempre entre o observador e um processo. Caso um processo precise enviar uma mensagem a outro, essa mensagem seria enviada ao observador que redirecionaria a mensagem ao processo em questão. Essa solução resolveria o problema de comunicação entre os processos, mas poderia representar um gargalo no sistema. Esse é um dos pontos em aberto desse trabalho.

2.2 Contribuições Desse Trabalho

Com a finalização da implementação proposta nesse trabalho, uma série de ferramentas foram concluídas para a inserção de um código de simulação distribuída sobre o protocolo *Rollback* Solidário.

Os serviços de criação e despacho dos processos para seus hosts de destinos foram concluídos, assim como os processos de retirada de eventos da lista de eventos para tratamento. A inserção de um evento na lista de eventos futuros em tempo de execução, assim como a sinalização de recebimento de uma mensagem *straggler* também foi concluída. Também estão finalizados os métodos para retorno à um tempo anterior da simulação, caso ocorra um *rollback*.

No processo observador, todo o processo de recebimento de *checkpoint*, detecção de corte consistente com base na matriz de dependência, eleição da linha de

retorno e envio das mensagens de *rollback* com a linha de recuperação para os processos está concluída.

2.3 Trabalhos Futuros

Com o fim da implementação desse trabalho, alguns novos pontos podem ser considerados para dar continuidade a esse projeto:

- Criação de um método eficiente de envio de mensagens *multicast* para os processos;
- Solução do problema de envio de mensagens para processos fora da mesma rede. A solução proposta de centralizar o tráfego de mensagens pode ser analizada como uma alternatica para o caso;