Simulação de Sistemas Distribuídos em Cenários com Defeitos

Renata de Moraes Trindade[1] Marinho Pilla Barcellos[1,2] Ingrid Jansch-Pôrto[1]

trindade@inf.ufrgs.br marinho@exatas.unisinos.br ingrid@inf.ufrgs.br

[1] Pós-Graduação em Ciência da Computação – UFRGS
[2] Programa Interdisciplinar de Pós-Graduação em Computação Aplicada – UNISINOS

1. Introdução

Um sistema distribuído (SD) é definido como um conjunto de vários computadores autônomos que não compartilham relógio nem memória, estão separados geograficamente e trocam informações através de mensagens sobre a rede de comunicação que os interliga [1]. Um SD deve lidar com várias fontes de indeterminismo: programas são executados em velocidades diferentes, utilizando um número qualquer de processadores, que estão distribuídos sobre uma topologia de rede desconhecida e com tempos de propagação e ordenamento de mensagens imprevisíveis [2]. Portanto, desenvolver protocolos distribuídos é uma atividade complexa. Além disso, os mecanismos para detectar e mascarar defeitos representam tipicamente uma grande parcela do desenvolvimento desses protocolos. Assim, testar, depurar e validar tais protocolos são tarefas que ainda apresentam desafios.

Simuladores contribuem para a solução destes problemas, pois oferecem um ambiente controlado para validar o comportamento de protocolos existentes, fornecem infra-estrutura para desenvolvimento de novos protocolos e oportunizam o estudo de suas interações. Entretanto, devido à complexidade dos sistemas distribuídos, ainda existem poucas iniciativas na área de simulação nesse contexto, notadamente com suporte adequado para o tratamento de situações com defeitos.

O Network Simulator versão 2 – ***ns***-2 [3] é uma ferramenta acadêmica iniciada em 1989 na Universidade da Califórnia em Berkeley. Em 1995, seu desenvolvimento foi apoiado pelo DARPA através do projeto VINT (Virtual InterNetwork Testbed [4]), envolvendo USC/ISI, Xerox PARC, LBNL, e UC Berkeley. Esta ferramenta tem sido amplamente utilizada por pesquisadores da área de redes, para estudo de políticas de filas, controle de congestionamento, desempenho de protocolos, protocolos de *multicast* confiável, entre outros. Além disso, tem código aberto e pode ser estendido para aplicações específicas.

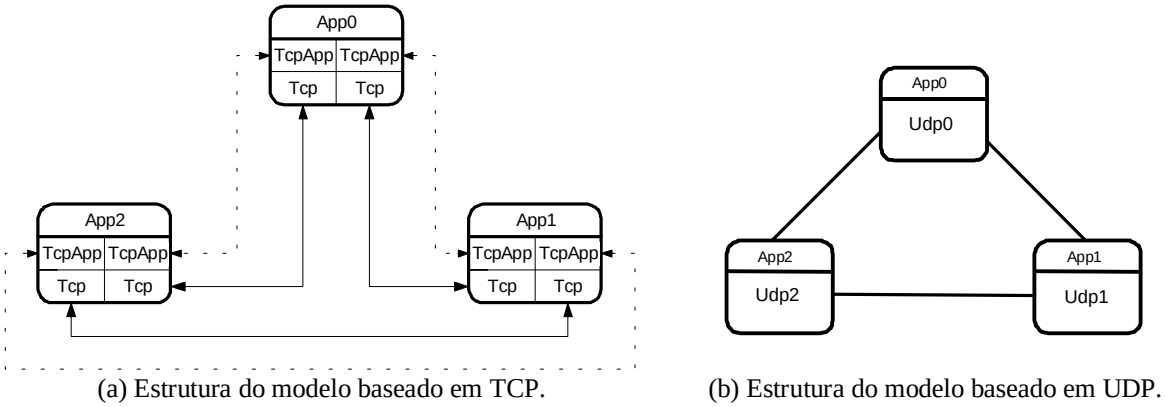
Este trabalho visa explorar a viabilidade de uso do ***ns*** como ambiente para simulação de SDs, particularmente em cenários sujeitos à ocorrência de defeitos. No presente artigo, são tratados apenas defeitos de colapso (*crash failures*); mas prevê-se a extensão da solução para outros tipos de defeitos.

2. Simulação de sistemas distribuídos no ns

Neste trabalho, dependendo do protocolo de transporte utilizado, são considerados dois modelos distintos para simulação, baseados: no protocolo TCP (confiável) ou no protocolo UDP (não confiável). Essas duas versões, modeladas em um sistema assíncrono [2], serão explicadas a seguir. Por premissa, cada nodo contém somente uma aplicação (processo que está na camada de aplicação).

No **modelo baseado em TCP**, o transporte de dados é feito por agentes TCP. Em um sistema com *n* nodos, cada nodo tem até *n-1* agentes TCP: um para cada conexão. Cada agente TCP é associado a um componente da camada de aplicação, chamado **`TcpApp`**, que é encarregado de fazer a emulação da transmissão de dados entre as aplicações. Além disso, em cada instância da aplicação (**`Appn`**), é mantida uma tabela de conexões, onde são armazenadas

todas as aplicações às quais ela está conectada, juntamente com a **`TcpApp`** encarregada de transmitir dados para a aplicação-destino. A Figura 1(a) mostra três nodos que trocam dados entre si, a distribuição dos agentes e aplicações dentro de cada nodo, e suas conexões. **`Appn`** representa um processo da aplicação distribuída, **`TcpApp`** é a classe encarregada de enviar os dados e TCP é o agente de transporte.



(a) Estrutura do modelo baseado em TCP. (b) Estrutura do modelo baseado em UDP.

Figura 2: Estruturas dos modelos.

Nesse modelo, a comunicação é bidirecional e feita por meio de *unicast*. O agente TCP usado é um agente bidirecional do ***ns*** chamado **`FullTcp`**. Para a transmissão de dados, estes percorrem o seguinte fluxo: a aplicação (por exemplo, **`App0`**) passa os dados à sua **`TcpApp`**, que os encaminha ao agente TCP. Este, por sua vez, os envia para o agente TCP da aplicação a qual está conectado. Quando um agente TCP recebe dados, ele os entrega à **`TcpApp`**, que os repassa à aplicação destinatária (**`App1`**).

O **modelo baseado em UDP** emprega agentes UDP para o transporte de dados; cada nodo tem somente um agente UDP que envia dados para destinatários. A comunicação não é confiável, mas tem a vantagem de oferecer também *multicast*, ao contrário do modelo baseado em TCP. A Figura 1(b) mostra como o agente UDP e a aplicação estão distribuídos dentro do nodo, nesse modelo.

No caso de *multicast*, cada agente UDP envia dados para o endereço de um grupo. Na Figura 1(b), por exemplo, **`App0`** (através de seu agente **`Udp0`**) envia dados para um grupo que é composto por **`Udp1`** e **`Udp2`**. O fluxo de dados é o seguinte: a aplicação (**`App0`**) passa os dados para seu agente UDP (**`Udp0`**) que, por sua vez, os envia para os agentes UDP de seu grupo-destino. Quando um agente UDP recebe dados, repassa-os para sua aplicação.

3. Simulações em cenários com defeitos

Para as atividades de simulação de defeitos em SDs, foram consideradas as seguintes definições: ocorre um defeito de **colapso (*crash*) em um processo**, quando este pára prematuramente e não faz mais nada a partir desse ponto, portanto não produz resultados incorretos. Defeito de **colapso em *link*** ocorre quando ele pára de transportar mensagens [5].

Verificou-se que o ***ns*** não oferece possibilidade direta para simular defeitos de nodo, mas apenas defeitos de *link*. Para estes, há um comando específico, cujos parâmetros definem o tipo de operação (rompimento/restabelecimento do *link*), identificação do *link* e o momento da ocorrência do defeito. Pode-se configurar defeitos permanentes ou temporários. O manual do ***ns*** indica a possibilidade de simular defeitos de nodos, utilizando o comando para defeito de *link*, identificando um nodo ao invés do *link*, dentre os parâmetros. Entretanto, a semântica desse comando indica que todos os *links* desse nodo serão "rompidos", o que impede o controle sobre o estado interno do nodo e de seus componentes. Isto é especialmente

problemático em variações tais como *crash-recover* ou casos de perda de conteúdo (*amnesia*).

Para simulação de defeitos de colapso em processos, foi criado um mecanismo que possibilita a expansão do modelo de defeitos. Para as versões TCP e UDP, foram desenvolvidas estruturas diferenciadas, apresentadas a seguir.

No modelo baseado em TCP, ao enviar dados, uma aplicação os entrega para **`TcpApp`**, que os repassa ao **`FullTcp`**, o qual os envia ao **`TcpApp`** remoto. Na recepção, o dado percorre o caminho inverso até atingir a aplicação (**`FullTcp`** → **`TcpApp`** → **`App`**). Portanto, o elemento mais baixo nas camadas atravessadas pelos dados, nesse modelo, é o agente **`FullTcp`**. A partir desta constatação, decidiu-se desenvolver uma classe derivada da classe **`FullTcpAgent`**, na qual os métodos que fazem o envio e a recepção dos dados foram reescritos para simular defeitos (**`sendmsg`** e **`recv`**, respectivamente).

A estrutura para simulação de defeitos, na versão baseada em UDP, diferencia-se devido aos elementos componentes dos nodos. Em cada nodo UDP, existe uma aplicação e somente um agente UDP: esse agente UDP é de uma nova classe – **`UdpData`** – derivada da classe **`UdpAgent`**. Sendo assim, o elemento da camada mais inferior é o agente **`UdpData`**. Assim, foi criada uma classe denominada **`UdpDataFail`**, derivada de **`UdpData`**, na qual os métodos de envio e recepção dos dados (**`send`** e **`recv)`** foram rescritos.

A Figura 2 apresenta as classes **`UdpDataFail`**, **`FullTcpFail`** e sua hierarquia em UML (Unified Modeling Language [6]), como será explicada a seguir. O método **`command`** é chamado quando é executado um comando do *script* da simulação OTcl para uma determinada classe. Por exemplo, assumindo-se que "**`tcp`**" é uma instância da classe **`FullTcpFail`**, o comando **`$tcp crash-failure`** atribui *true* à variável **`crash`**, indicando que existe defeito de colapso nesse agente/aplicação. No método **`recv`**, as ações dependem do valor da variável **`crash.`** Em *false*, indica que o nodo não está com defeito, nesse caso, é realizada a chamada para o método **`recv`** de sua superclasse. Se houver defeito no nodo (se **`crash=true`**), o pacote recebido será eliminado. Portanto, na recepção de dados, essa função verifica o valor de **`crash`** e, dependendo desse valor, passa o pacote para a classe **`FullTcpAgent`** (ou **`UdpAgent`**) ou o elimina. No método **`sendmsg/send`**, o procedimento é semelhante ao feito no método **`recv.`** Se o valor da variável **`crash`** for igual a *false*, então é chamado o método **`sendmsg/send`** de sua superclasse. Senão, se o agente está com defeito (valor de **`crash=true`**) os dados não são enviados para o agente **`FullTcp`** e, portanto, a aplicação-destino não irá recebê-los.

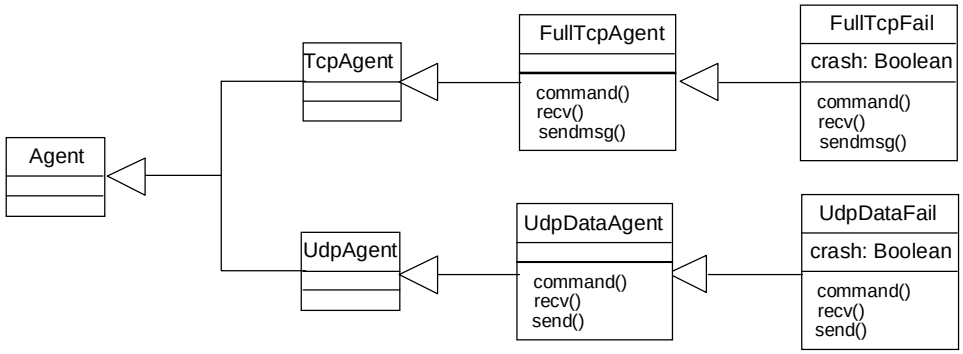


Figura 2: Modelo simplificado de classes.

4. Estudo de casos

Para exemplificar os modelos de simulação apresentados, foram implementados, no ***ns***, os seguintes algoritmos distribuídos baseados nos algoritmos propostos por Lynch[2]: (a) **algoritmo de eleição em anel assíncrono**: desenvolvido para solucionar o problema de

eleição de um processo líder em um anel unidirecional em uma rede assíncrona. Esse algoritmo é chamado *AsynchLCR*; (b) **algoritmo de inundação assíncrono**: soluciona o problema de eleição de líder em qualquer tipo de topologia; (c) **algoritmo para solucionar do problema de concordância em sistemas assíncronos**: atinge concordância mesmo na presença de defeitos de colapso em processos (chamado de *BenOr*).

Estes algoritmos foram implementados utilizando ambos modelos baseados em TCP e UDP. Nota-se que a estrutura básica para simulação de SDs nos diferentes casos é semelhante, os casos diferenciam-se pelos algoritmos distribuídos escolhidos.

Foi desenvolvido um arquivo de *script* de simulação que serve como padrão para desenvolvimento de qualquer aplicação que utiliza TCP, em qualquer tipo de topologia física. Este *script* foi feito para facilitar a criação e o estabelecimento de conexões dos agentes e a criação de aplicações contidas nos nodos do sistema. Com o uso desse arquivo, é possível criar todos os nodos, agentes e conexões do experimento. Além desse arquivo *script* padrão, foi criado outro arquivo que facilita a simulação de defeitos. Um de seus principais procedimentos é o **`crash_failure <n_node> <failure_node>`**, onde **`<n_node>`** é a quantidade de nodos e **`<failure_node>`** é o número do nodo onde deve ocorrer o defeito. Com esse procedimento, são feitas chamadas ao **`comando $tcp crash-failure`** para todos os agentes **`FullTcpFail`** do nodo **`<failure_node>`**, que é responsável por simular defeito de colapso em toda a aplicação do nodo.

A simulação desses algoritmos no ***ns*** produziu resultados funcionalmente corretos. Mas apenas o algoritmo *BenOr* prevê a ocorrência de defeitos. Nesta implementação, foram realizados vários experimentos funcionais, inserindo-se defeitos em vários momentos da simulação. Foi constatado que, também neste cenário com defeitos, respeitadas as hipóteses adotadas no desenvolvimento do algoritmo, os resultados gerados também são corretos.

5. Conclusões

O presente trabalho tem como objetivo estudar a adequação do simulador de redes ***ns*** como instrumento para a verificação e desenvolvimento de protocolos em SDs. Além da riqueza de funções e aplicações, a ferramenta foi escolhida por suas características acadêmicas: código aberto e possibilitar o desenvolvimento de extensões.

Foram identificados os mecanismos necessários para a simulação de sistemas com topologias variadas; comenta-se a implementação de uma rede totalmente interconectada, cuja rede de comunicação permite a opção entre protocolos UDP e TCP. Foi proposta uma solução para simulação de defeitos de colapso, seu desenvolvimento e resultados obtidos com casos exemplo. Além de testes adicionais em diferentes arquiteturas e variações de implementação, o prosseguimento deste trabalho inclui a expansão do modelo de defeitos suportado pelo simulador. O código-fonte das alterações no ***ns*** e a implementação dos algoritmos será disponibilizada para *download* e livremente distribuída segundo a licença GPL.

Referências Bibliográficas

[1] Jalote, P. Fault Tolerance in Distributed Systems. New Jersey: Prentice Hall, 1994.

[2] Lynch, N. Distributed Algorithms. San Francisco: Morgan Kaufmann Publ., 1996.

[3] Network Simulator – NS-2: http://www.isi.edu/nsnam/ns/index.html.

[4] VINT – Virtual InterNetwork Testbed: http://www.isi.edu/nsnam/ vint/index.html.

[5] Hadzilacos, V.; Toueg, S. Fault-Tolerant Broadcasts and Related Problems. Distributed Systems. New York: Addison-Wesley, 1995. chap.5, p.99-102.

[6] Booch, G; Rumbaugh, J; Jacobson, I. The Unified Modeling Language User Guide. New York: Addison-Wesley, 1999.