Quebrando a Barreira entre Simulação e Experimentação Prática em Redes de Computadores*

Hisham H. Muhammad **Guilherme B. Bedin**
Giovani Facchini **Marinho P. Barcellos**

{hisham,bedin,giovani,marinho}@exatas.unisinos.br

[1] PIPCA - Programa de Pós-Graduação em Computação Aplicada
UNISINOS - Universidade do Vale do Rio dos Sinos
Av. Unisinos, 950. São Leopoldo, RS. CEP 93022-000

Resumo. Simulação e experimentação são duas técnicas complementares para a realização de avaliações de desempenho, cada uma possuindo características e vantagens opostas. Enquanto uma permite total controle e abstração no experimento, outra proporciona maior detalhamento e realismo. Embora idealmente realizar ambas seja desejado, é complicado dispender os recursos necessários para realizar o desenvolvimento de uma avaliação de desempenho duas vezes, uma sobre um simulador e outra sobre um ambiente real. Neste artigo uma nova abordagem é explorada através de uma ferramenta, o Simmcast Testbed, que permite que a partir de um código único, experimentos sejam executados tanto em modo de simulação como em modo de experimentação. Um exemplo didático de uso é discutido em detalhes e a correlação dos resultados simulados e experimentais é apresentada.

Abstract. Simulation and experimentation are two complementary techniques for performance evaluation, each one of them having opposite characteristics and advantages. While one allows total control and abstraction in the experiment, the other provides greater detail and realism. Though ideally it would be desirable to perform both, it is hard to direct the required efforts to develop a performance evaluation twice, once over a simulator and then again on a real network. In this paper a new approach is explored through a tool called Simmcast Testbed, which allows one to execute, from a single codebase, experiments both in simulation and experimentation mode. A didactic example is discussed in detail, and the correlation of the simulated and experimental results is presented.

Palavras-chave: Aspectos de desempenho, escalabilidade e confiabilidade, Tolerância a falhas, Simulação

1. Introdução

Examinando-se artigos nas principais conferências na área de redes de computadores, percebe-se que boa parte emprega alguma forma de avaliação de desempenho de

*este trabalho foi desenvolvido com auxílio financeiro do CNPq.

protocolos ou algoritmos. Três formas principais de avaliação são verificadas: *avaliação analítica*, *simulação* e *experimentação*. Avaliação analítica é muito útil para demonstrar propriedades gerais do protocolo, investigando o desempenho mediante variações nos parâmetros de entrada. Entretanto, funciona bem apenas para modelos simples, passíveis de uma modelagem em termos de um conjunto de equações.

Simulação permite um grau variável de abstração, do simples ao detalhado, dependendo do modelo de simulação construído e do código correspondente. Conforme indicado em [1], simulação tem o potencial de permitir o incremento gradual de detalhe, de forma que o processo resulte em um protocolo executando sobre uma "rede emulada", pronto para ser movido para uma rede real.

Experimentação está baseada em um conjunto de execuções de teste de um protocolo sobre uma rede real, e portanto é capaz de produzir resultados mais confiáveis. Não obstante, por causa da grande influência da topologia, dos sistemas e das configurações envolvidas (rede não controlada sujeita à tráfego de fundo, etc.), os resultados coletados são específicos de uma configuração, e são difíceis de se reproduzir por outros pesquisadores. A logística exigida na preparação dificulta a realização de experimentos, particularmente quando envolvem redes com múltiplas organizações. Experimentação, naturalmente, exige a implementação do protocolo, ou no mínimo de um protótipo do mesmo. Cada uma dessas metodologias possui seus méritos relativos, com seus prós e contras. Não existe uma que seja aplicada igualmente bem a todas as situações de avaliação de desempenho.

Embora na área de redes de computadores simulação discreta seja uma metodologia bastante utilizada na avaliação de desempenho, os resultados produzidos pela mesma, isoladamente, não são suficientemente confiáveis. Limitações ou erros comuns no emprego de simulação, como aqueles indicados em [4] e [3], podem induzir distorções na avaliação. O mesmo poderia ocorrer com avaliação analítica e experimentação. Particularmente, neste último, distorções podem ocorrer por causa de condições imprevistas, em função de tráfego alheio, configuração incorreta dos computadores envolvidos ou equipamento de rede. Portanto, seria recomendável utilizar as três formas de avaliação **complementarmente**, como parte do processo de entendimento do funcionamento de um protocolo, e obtenção de medidas de desempenho para o mesmo em função de seus principais parâmetros e configurações de rede desejadas. Por uma questão prática, é provável que dois métodos sejam suficientes; quais dois dependerá principalmente do problema analisado e das ferramentas disponíveis.

Uma alternativa atraente para muitos casos seria o uso de simulação em conjunto com experimentação. Na maioria dos casos, simulação ou experimentação são empregados isoladamente. Isso ocorre devido ao esforço de implementação asso-

ciado a cada um destes métodos. Nos raros casos em que o pesquisador se dá ao trabalho de validar sua proposta com simulação e experimentação, estas metodologias são usadas de forma estanque, uma após a outra. Isto ocorre, também, porque o código de simulação é independente e diferente do código do protótipo utilizado na experimentação.

Seria desejável que simulação e experimentação pudessem ser parte de um mesmo conjunto, integrado, de maneira que o pesquisador pudesse intercambiar simulação e experimentação quase sem esforço (*seamlessly*). Naturalmente, nem todo tipo de trabalho sobre protocolos permite esse tipo de situação; por exemplo, é preciso um suporte de simulação que ofereça diferentes níveis de abstração, e que seja capaz mesmo de emular as camadas subjacentes. Neste artigo, é proposta uma extensão ao *framework* de simulação Simmcast ([1, 8]) que permite o uso de simulação em conjunto com experimentação para protocolos em nível de aplicação.

O restante do artigo está estruturado da seguinte forma: a Seção 2 apresenta o Simmcast Testbed, incluindo sua interface e forma de uso. A Seção 3 descreve um modelo simulado de algoritmo distribuído, adotado como Estudo de Caso. Este algoritmo é exercitado na Seção 4, com o objetivo de obter resultados experimentais e de simulação. Medindo-se a correlação entre os resultados de simulação e os resultados reais, avalia-se o poder da abordagem proposta. Comentários finais e trabalhos futuros são parte da Seção 5.

2. Simmcast Testbed

O Simmcast Testbed é uma interface de programação (API, *Application Programming Interface*) para a linguagem Java, que isola o programador tanto das peculiaridades do ambiente de simulação como dos detalhes das bibliotecas de rede. Esta API compreende recursos de comunicação e um modelo de *threads*, sendo composta por duas implementações, de acordo com o *back-end* utilizado: ou o *framework* Simmcast ou as APIs do Java para programação em rede e *threads*.

O Testbed está baseado no Simmcast, um *framework* de simulação de protocolos e sistemas distribuídos. A seguir, apresenta-se brevemente o Simmcast, enfatizando o papel do mesmo como *back-end* de simulação. Após, a API do Simmcast Testbed é apresentada em detalhes, discutindo-se a sua interface e a implementações, e por fim o uso do Simmcast Testbed.

2.1. Simmcast como *back-end* de simulação

O Simmcast é um projeto de pesquisa que deu origem ao framework para simulação de redes de mesmo nome, apresentado anteriormente ([1, 8, 9]). O Simmcast Testbed,

ao mesmo tempo em que está inserido no contexto do projeto de pesquisa Simmcast, faz uso deste *framework* de simulação como um dos *back-ends* possíveis para a sua execução. As principais características do Simmcast são uma arquitetura modular e um modelo de simulação baseado em processos, conforme explicado a seguir.

Na arquitetura do Simmcast, as classes correspondem a "blocos de construção" (*building blocks*) que são dinamicamente conectados de modo a construir o ambiente de simulação. Estações correspondem a objetos `HostNode`, roteadores a objetos `RouterNode` (ambos derivados de `Node`, que representa uma entidade abstrata).

O modelo de simulação baseado em processos mostra-se bastante adequado para a simulação de sistemas computacionais, uma vez que cada processo corresponde diretamente a uma *thread* de execução do programa que está sendo simulado. No Simmcast, cada processo ou *thread* é mapeado como um objeto `NodeThread`. Ao permitir que objetos `Node` (e por conseguinte, `HostNode` e `RouterNode`) constituam-se de um ou mais objetos `NodeThread`, protocolos *multi-threaded* podem ser projetados de forma natural.

Estas duas características fazem com que a API do Simmcast seja bastante realística, similar às APIs reais de comunicação e de *threads* da linguagem Java, ao mesmo tempo que permite simulações mais abstratas, através das entidades de mais alto nível como `Node`. Esta proximidade intencional das APIs foi um fator decisivo no desenvolvimento do Simmcast Testbed. Isto permitiu que o projeto desta nova API fosse restrito pelas características de um ambiente de execução real, e não limitada por simplificações do ambiente de simulação.

2.2. A API Simmcast Testbed

O Simmcast Testbed é uma API que possui duas implementações: `Testbed-Sim`, tendo o *framework* Simmcast como *back-end*, e `Testbed-Exp`, tendo como *back-ends* as APIs `java.net` e `java.lang.Thread`. Ambas são implementadas como um pacote Java denominado `simmcast.testbed`.

O modelo de *threads* do Simmcast Testbed segue o modelo da linguagem Java, e é composto de duas classes: `Program` e `ProgramThread`. A classe `ProgramThread` corresponde a uma *thread* Java, e de fato, o conjunto de métodos oferecido é bastante similar. A classe `Program`, por sua vez, corresponde a uma instância independente de um programa, incluindo o método principal `main` a partir do qual as demais *threads* são iniciadas. Considere, por exemplo, uma aplicação cliente-servidor onde o servidor possui múltiplas threads para tratamento concorrente de requisições. Neste caso, cliente e servidor corresponderiam a duas instâncias inde-

pendentes de `Program`, enquanto as threads do servidor seriam múltiplas instâncias de `ProgramThread`. Outro exemplo de uso de threads no Simmcast consiste na estruturação de um protocolo complexo em threads de comportamento síncrono, separando em um nodo os papéis de envio e recepção de pacotes; neste caso haveria uma instância de `Program` e duas de `ProgramThread`.

As primitivas de envio e recebimento de dados do Simmcast Testbed estão num nível de abstração intermediário entre o modelos do Simmcast e de Java. No Simmcast Testbed, são disponibilizadas, diretamente na classe `ProgramThread`, três primitivas para envio e recebimento de dados: `send()`, `receive()` e `receiveMulticast()`. O método `send()` recebe um vetor de bytes como parâmetro e os métodos `receive()` e `receiveMulticast()` retornam um objeto `DataPacket` contendo o vetor de bytes recebido.

O gerenciamento de *sockets* é feito no `Testbed-Exp` de forma implícita: ao utilizar uma primitiva de envio ou recebimento de dados para um endereço não especificado previamente, um novo *socket* é aberto internamente. Outra abstração importante é a gerência de grupos multicast: na implementação `Testbed-Exp`, ao enviar para um novo grupo, a requisição de `joinGroup()` é realizada automaticamente; na implementação `Testbed-Sim`, o bloco primitivo `Group` do Simmcast é utilizado. No caso de `Testbed-Exp`, as transmissões de dados ocorrem por datagramas.

Enquanto em Java endereços IP são codificados através de objetos `InetAddress`, no simulador Simmcast nodos e grupos multicast são identificados simplesmente através de identificadores numéricos. No Simmcast Testbed, a classe `NetworkAddress` abstrai o endereçamento de rede. Métodos como `getLocalHost()` são disponibilizados de modo que os endereços sejam sempre especificados na forma de objetos `NetworkAddress`.

Temporizadores assíncronos são um recurso comum utilizado na descrição de protocolos. O Simmcast possui um recurso de *callback* através do método `onTimer()`, que simplifica a especificação de temporizadores. Esse recurso foi replicado no Simmcast Testbed.

Devido a estas características, a API Simmcast Testbed possui uma correspondência bastante direta tanto com o modelo de programação de redes e *threads* em Java, como com o modelo de programação do Simmcast. De fato, esta semelhança pode ser observada na Tabela 1, que resume e compara os métodos.

Simmcast Testbed	Simmcast	Java
ProgramThread	NodeThread	Thread
Program	HostNode	*programa principal*
DataPacket	TransportPacket	DatagramPacket
NetworkAddress	Node.networkId (int)	InetAddress
Clock.getDate()	Network.simulationTime()	System.currentTimeMillis()
ProgramThread.send()	Node.send()	DatagramSocket.send()
ProgramThread.receive()	Node.receive()	DatagramSocket.receive()
ProgramThread.receiveMulticast()	Node.receive()	MulticastSocket.receive()

Tabela 1: Correspondência entre métodos e classes do Simmcast Testbed e os equivalentes no simulador Simmcast e na API da linguagem Java.

2.3. Uso do Simmcast Testbed

A escolha do modo de execução (simulado ou real) se dá em tempo de execução, ao especificar à *Java Virtual Machine* (JVM) qual cópia do pacote `simmcast.testbed` deve ser utilizada. Desta forma, é possível alternar entre os modos `Testbed-Sim` e `Testbed-Exp` sem a necessidade de recompilação.

Na execução real, cada instância de `Program` corresponde a uma JVM independente, possivelmente executando em computadores distintos. Na execução simulada, uma única instância do simulador Simmcast irá criar todos as instâncias de `Program` e executá-las em uma mesma JVM. Devido a este fato, a forma como experimentos são iniciados é diferente nos dois modos, e deve ser preparada em separado. Para a inicialização do modo `Testbed-Exp`, o Simmcast Testbed oferece uma classe `Main`, que possui o método estático `main` e recebe uma classe `Program` como parâmetro. No modo `Testbed-Sim`, a inicialização dos objetos `Program` é feita através do arquivo de descrição de simulação do Simmcast, que faz uso do recurso de carga dinâmica de classes de Java. Uma vez realizada a inicialização, todo o código utilizado no experimento nos modos `Testbed-Exp` e `Testbed-Sim` é idêntico.

Ao utilizar o Simmcast Testbed, uma série de cuidados devem ser tomados para assegurar a portabilidade da execução nos dois modos. O principal destes cuidados é não utilizar diretamente as APIs de rede e *threads* de Java, ou as APIs do Simmcast, intercaladas com código que utiliza o Simmcast Testbed. Caso isso ocorra, é possível que o código resultante funcione apenas em um dos ambientes. Outros cuidados importantes referem-se às peculiaridades dos ambientes de execução real e simulado. Enquanto o ambiente real permite o uso de variáveis estáticas, já que

cada estação executa o seu programa em uma máquina virtual separada, no modo de simulação todos os objetos `Program` compartilharão erroneamente a variável.

Já o compartilhamento intencional de variáveis, embora seja uma uma técnica comum em simulação, consiste em um problema de portabilidade para programas desenvolvidos inicialmente no modo `Testbed-Sim` e depois executados no modo `Testbed-Exp`. Variáveis globais consistem uma vantagem importante de simulações, pois são usadas tipicamente para declaração de parâmetros de entrada globais à simulação, tal como para coleta global de resultados.

Uma das principais diferenças entre a execução simulada e a real decorre da ausência, na segunda, de um relógio global único (afinal, os nodos compõem um sistema distribuído). Na simulação discreta, existe uma variável interna da máquina de simulação para representar o relógio da simulação, sendo este comum à toda simulação. Um relógio comum é uma importante abstração no que se refere a medições de tempo e ao ordenamento global de eventos. A seguir, isto é ilustrado através de dois exemplos. Primeiro, sejam dois nodos, A e B, para medir o tempo de propagação unidirecional de A até B, é suficiente registrar o tempo de envio em A, t_A, o tempo de chegada em B, t_B, e a latência desejada será obtida por um $t_B - t_A$. Em um sistema típico, tal medição só pode ser realizada de forma precisa através de um esquema externo para sincronização física dos relógios de A e B, como por exemplo usando receptores GPS. No segundo exemplo, o relógio único permite que nodos registrem em uma console ou arquivo de *log* comum eventos relevantes e o tempo em que os mesmos ocorreram. Ordenando-se esse registro por tempo, o entendimento da execução distribuída é facilitado, utilizando-se para isso por exemplo uma ferramenta de visualização.

3. Estudo de caso

Nesta seção, será apresentado em um estudo de caso que demonstra o uso do Simmcast Testbed. Como o objetivo aqui consiste em apresentar a metodologia de desenvolvimento, será empregado um exemplo bastante simples, de modo que possa ser apresentado de forma completa. O exemplo empregado será o algoritmo para eleição de um nó líder em um anel unidirecional segundo o modelo temporal síncrono, conforme apresentado em [6]. Uma implementação deste algoritmo consta do conjunto de *demos* do Simmcast[1]. Através do Simmcast Testbed, este exemplo é adaptado de modo a ser executado tanto no simulador, como sobre a rede real.

[1]Disponível na página do projeto, `http://www.unisinos.br/~simmcast`.

3.1. O algoritmo de eleição

O algoritmo de eleição em anel unidirecional é um dos exemplos clássicos na área de sistemas distribuídos. A implementação original no Simmcast descreve o algoritmo de maneira bastante direta, utilizando parâmetros de configuração dos *links* de modo a considerar como premissa a existência de uma rede síncrona e confiável. Para fins de melhor visualização das rodadas do algoritmos, atrasos são inseridos antes do envio das mensagens.

Cada nodo é implementado como um objeto da classe `RingNode` (uma subclasse da classe `Node` do Simmcast) que contém uma única *thread* de execução, `RingElectionThread`, na qual o algoritmo é descrito. No arquivo de configuração, uma rede com topologia em anel é especificada e parâmetros como indicação do nodo vizinho são passados a cada um dos nodos. Cada nodo recebe ainda um identificador lógico único, a partir do qual o líder será escolhido.

3.2. Adaptação para o Simmcast Testbed

A preparação para execução do algoritmo no Simmcast Testbed compõe-se de duas etapas: primeiro, é necessário adaptar o programa para a API do Simmcast Testbed; posteriormente, é preciso adaptar o código de modo a remover premissas inerentes ao modelo de simulação. A primeira etapa é direta. Conforme apresentado na Seção 2.2, a API é intencionalmente similar tanto à API do simulador como à API da linguagem Java, de modo a facilitar conversões em ambos os sentidos. Durante a realização desta etapa, o programa pode ser testado sobre o simulador utilizando o modo de execução `Testbed-Sim`.

A segunda etapa consiste em remover do experimento simplificações da simulação que impediriam o seu funcionamento em uma rede real, incluindo possíveis medições de intervalos de tempo baseadas na premissa de um relógio de simulação global. Esta etapa exige uma análise das características do experimento em questão. No exemplo simples apresentado aqui, duas modificações bastam para permitir a execução do experimento sobre a rede: garantir a entrega das mensagens e a sincronia das rodadas. Feito isto, o programa poderá ser executado tanto no modo `Testbed-Sim` como no modo `Testbed-Exp`.

A transmissão confiável de mensagens entre dois vizinhos é implementada como uma camada subjacente ao programa em si, mantendo o algoritmo de eleição inalterado. Duas novas *threads,* denominadas `TxM` e `RxM`, são adicionadas ao `RingNode` (agora uma subclasse da classe `Program` do Simmcast Testbed), para implementar um *buffer* de envio e um de recebimento, respectivamente. Em outras palavras, elas são responsáveis por gerenciar retransmissões e envio de pacotes de

```
boolean leader = false;
Integer value = new Integer(ringNode.logicalId);
send(new Packet(source, dest, type, size, value));
for (int round = 1; round <= ringNode.N; round++) {
  Packet pkt = (Packet) receive(timeout);
   if (pkt == null)
     continue;
  value = (Integer)(pkt.getData());
   if (value.intValue() < ringNode.logicalId) {
    sleep(discardPacketRoundDelay);
   } else if (value.intValue() == ringNode.logicalId) {
    leader = true;
    sleep(leaderDelay);
   } else {
    sleep(forwardPackeRoundDelay);
    send(new Packet(source, dest, type, size, value));
   }
}
```

(a) Implementação da `RingElectionThread` no simulador Simmcast.

```
boolean leader = false;
Integer value = new Integer(ringNode.logicalId);
byte[] data = Serializer.serialize(value);
txm.send(this, data, data.length, dest, port);
for (int round = 1; round <= ringNode.N; round++) {
  DataPacket pkt = rxm.receive(this, port, timeout, data.length);
   if (pkt == null)
     continue;
  value = (Integer)(Serializer.rebuild(pkt.getData()));
   if (value.intValue() < ringNode.logicalId) {
    sleep(discardPacketRoundDelay);
   } else if (value.intValue() == ringNode.logicalId) {
    leader = true;
    sleep(leaderDelay);
   } else {
    sleep(forwardPacketRoundDelay);
    data = Serializer.serialize(value);
    txm.send(this, data, data.length, dest, port);
   }
}
```

(b) Adaptação da `RingElectionThread` para o Simmcast Testbed

Figura 1: Comparação entre o código original para o simulador e o código adaptado para execução tanto no simulador quanto sobre a rede real.

confirmação. A única alteração na `RingNodeThread`, além de conversões sintáticas de API, é passar a realizar envios e recebimentos através de `TxM` e `RxM`. A Figura 1 compara o laço de execução principal das versões original e adaptada do algoritmo. Note que uma classe acessória, `Serializer`, realiza a serialização e reconstrução

de objetos Java para a transmissão em datagramas (no simulador, objetos Java podem ser inseridos diretamente em pacotes).

O esquema de sincronização de rodadas empregado foi bastante simplista, de modo a manter a similaridade com a versão original do algoritmo apresentada na Figura 1(a). Além de um mecanismo simples para inicialização simultânea dos nodos (específico para o modo `Testbed-Exp`), os atrasos entre rodadas foram mantidos, de forma que o *buffer* de recebimento compense as variações no tempo de propagação dos pacotes na rede real.

4. Avaliação dos Resultados

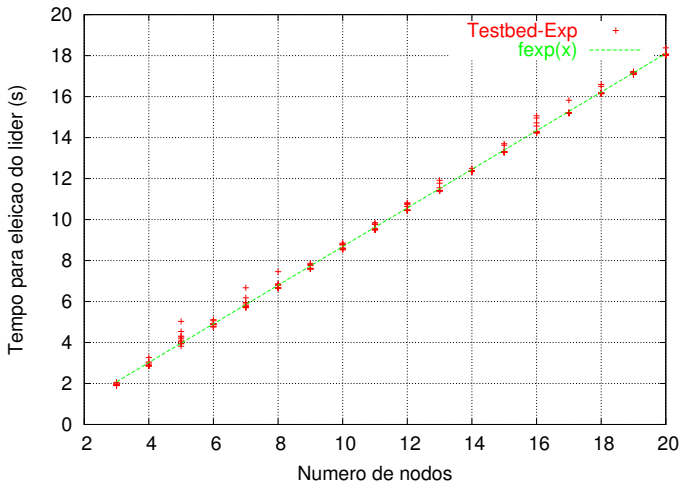
A adaptação do algoritmo de eleição em anel unidirecional para o Simmcast Testbed foi executada nos modos `Testbed-Sim` e `Testbed-Exp`. O algoritmo foi executado variando o número de nodos de 3 a 20. Para cada número de nodos, rodaram-se 10 iterações.

O experimento com `Testbed-Exp` foi executado em ambiente LAN com 20 PCs conectados a um *switch* 100Mbps. Para reproduzir este cenário, o arquivo de configuração do ambiente de simulação foi modificado de modo a descrever uma topologia em estrela, onde todos os `RingNodes` foram conectados a um nodo `Default-RouterNode`. As propriedades dos enlaces simulados foram configuradas de modo a espelhar a rede real, com banda de 100Mbps e latência descrita por uma distribuição normal de média 0,5ms e desvio padrão 0,1.

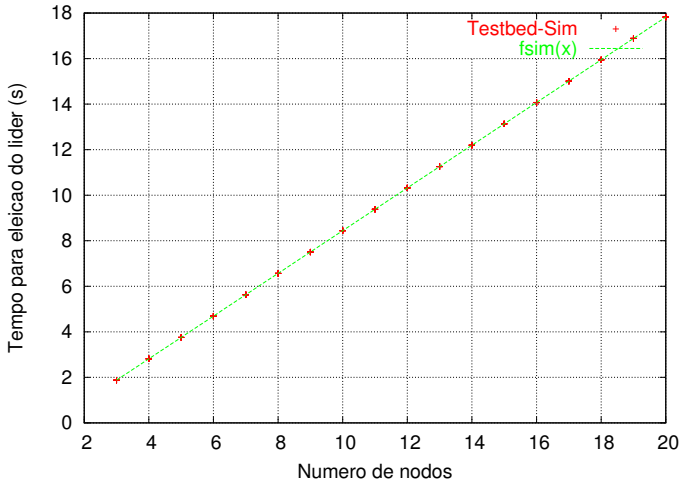
O resultado das execuções nos modos `Testbed-Exp` e `Testbed-Sim` é apresentado nas Figuras 2(a) e 2(b), respectivamente. Em ambos os casos são exibidos os pontos correspondentes a todas as iterações e o ajuste linear para cada modo de execução. A função de ajuste linear da execução no `Testbed-Exp` é $fexp(x) = -0.74606 + 0.942619x$, e o ajuste da execução no Testbed-Sim é dado por $fsim(x) = -0.937052 + 0.938019x$ (todos os valores em segundos).

O ajuste dos pontos a uma função tem como objetivo representar o comportamento do algoritmo sendo modelado. Ao comparar os ajustes obtidos entre as execuções em modo simulado e experimental, torna-se possível verificar se os comportamentos obtidos pela execução de um mesmo programa em ambos os modos são equivalentes.

As duas funções de ajuste são apresentadas em conjunto na Figura 3. Pode-se observar claramente que é possível obter valores bastante próximos nas execuções real e simulada e, principalmente, que o comportamento do algoritmo nos dois modos são consistentes entre si à medida que a escala do experimento varia.



(a) Execução experimental



(b) Execução simulada

Figura 2: Resultados da execução do algoritmo de eleição em anel unidirecional

5. Conclusão

O Simmcast Testbed vem sendo utilizado no contexto do projeto Metropoa ([7]), na investigação de modelos de arquiteturas de jogos online em larga escala. O Testbed

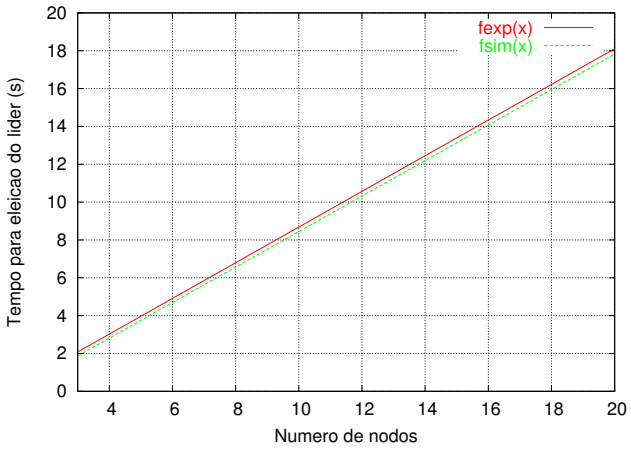


Figura 3: Comparação das funções de ajuste linear das execuções experimental e simulada

atua como um elo entre duas das principais técnicas empregadas para avaliação de desempenho nas áreas de redes de computadores e sistemas distribuídos: simulação e experimentação. O que foi apresentado neste artigo não é apenas uma ferramenta, mas uma metodologia para desenvolvimento em pesquisas nesta área, na qual o ambiente controlado de um simulador pode ser usado para a implementação e depuração, e a riqueza de detalhes de um ambiente de rede real pode ser usado na extração de resultados. A principal vantagem desta metodologia é a possibilidade de alternar entre os dois ambientes conforme necessário, sem a necessidade de portar o código novamente. Isto somente é possível devido ao fato de que o Simmcast Testbed é uma API "transicional", situada entre os níveis de abstração de um simulador e de uma biblioteca de redes.

A abordagem proposta neste artigo é inédita. A mesma não deve ser confundida com facilidades de "emulação de rede", como as oferecidas por certos simuladores. No VINT ns-2, emulação ([12]) significa permitir que uma simulação seja executada em uma máquina que irá gerar, como eventos em sua simulação, o envio de pacotes na rede verdadeira; conversamente, uma simulação em uma outra máquina pode teoricamente receber pacotes e interpretar o recebimento de tal pacote como um entrada (um evento) da simulação. Note que o relógio da simulação difere do relógio de tempo real, pois trata-se de duas simulações distintas, com relógios e eventos discretos independentes. Para NIST ([11]) e Delayline ([10]), emulação significa executar aplicações em um *testbed* composto por um conjunto de estações modificadas de forma a introduzir atrasos, perdas de pacotes e outras condições de rede desejáveis em um experimento.

Uma iniciativa que compartilha motivações similares a este trabalho é o Netbed ([16]). O Netbed é um ambiente que oferece um conjunto comum de abstrações para diferentes tipos de enlaces e nodos de forma a compor um ambiente que integra simulação, emulação e experimentação em rede real. O seu *design* generaliza recursos e mecanismos em abstrações comuns aplicáveis a uma variedade de realizações de emulação, simulação e experimentação. O Netbed se baseia no NS-2 e oferece aos usuários o acesso a um *testbed* físico e nodos distribuídos através de uma interface Web. O Netbed é diferente no sentido que não permite que o mesmo código do protocolo ou aplicação seja migrado transparentemente, tal como propiciado pelo Testbed Simmcast.

A abordagem aqui proposta pode ser comparada em relação aos métodos de simulação e experimentação, quando empregados de forma isolada. Individualmente, é esperado que cada um deles exija menor esforço de desenvolvimento do que utilizando o Simmcast Testbed. Sem usar o Testbed, a quantidade de trabalho necessário para "converter" uma simulação em um protótipo que funcione variará caso a caso, dependendo do problema alvo, da linguagem utilizada, e da forma com que o mesmo foi modelado na simulação. Quanto maior o nível de detalhe incluído na simulação, e permitido pelo simulador, menor deverá ser este esforço.

Em termos de facilidades, a API do Testbed impõe certas restrições quanto ao que pode ser feito/empregado pelo código do protocolo ou aplicação. Notavelmente, o único suporte atualmente existente é a mensagens enviadas via datagrama UDP. Ou seja, não há suporte nem a *streams* TCP nem a facilidades de comunicação de mais alto nível, como *Remote Method Invocation* (RMI). Algumas dessas restrições são necessárias para que a implementação rode apropriadamente em ambas plataformas, simulada e real.

Em termos de desempenho, a versão Exp do protocolo apresentará resultados inferiores em relação a uma "implementação nativa em Java", devido à sobrecarga da camada fina `Testbed-Exp` que implementa a API. Ainda não existem dados que demonstrem com segurança quão representativa é essa sobrecarga, mas baseados nas avaliações conduzidas, as estimativas iniciais são promissoras.

Referências

[1] M. P. BARCELLOS, H. MUHAMMAD, and A. DETSCH, "Simmcast: a Simulation Tool for Multicast Protocol Evaluation", XIX Simpósio Brasileiro de Redes de Computadores (SBRC 2001), Anais, SBC, Flps., 21-25 Maio 2001.

[2] L. BRESLAU et alli. "Advances in Network Simulation". In IEEE Computer, volume 33, n. 5, pp. 59-67, May 2000.

[3] J. BYERS, G. HORN, M. HANDLEY, M. LUBY, W. SHAVER, and L. VICISANO. "More Thoughts on Reference Simulations for Reliable Multicast Congestion Control Shemes". Notes from a meeting at Digital Fontain, August 8, 2000.

[4] R. JAIN. "The Art of Computer Systems Performance Analysis". John Wiley & Sons, 1991.

[5] I. KEIDAR, R. KHAZAN, N. LYNCH & A. SHVARTSMAN, "An Inheritance-Based Techique for Building Simulation Proofs Incrementally", ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 1, January 2002.

[6] N. LYNCH, "Distributed Algorithms", Morgan Kaufmann, San Francisco, 1996.

[7] METROPOA, Site do Projeto, `http://prav.unisinos.br/metropoa/metropoa2.php`

[8] H. H. MUHAMMAD, M. P. BARCELLOS, "Simulating Group Communication Protocols Through an Object -Oriented Framework", 35th Annual Simulation Symposium (SS2002), Proceedings, IEEE (New York), San Diego, 14-18 April 2002.

[9] H. H. MUHAMMAD, M. P. BARCELLOS, R. CASAIS, "Simulação de Roteamento na Avaliação de Protocolos Multicast e Sistemas Distribuídos de Grupo", I Workshop em Desempenho de Sistemas Computacionais e de Comunicação (Wperformance 2002), XXII Congresso da SBC 2002, Anais, SBC, Florianópolis, 18-19 Julho 2002

[10] D. B. INGHAM, G. D. PARRINGTON, "Delayline: A Wide-Area Network Emulation Tool", Computing Systems, v.7, n.3, 1994.

[11] NISTNET, Site do Projeto, `http://dns.antd.nist.gov/itg/nistnet/`, acessado em Novembro de 2003.

[12] NS-2, Site do Projeto, recurso com documentação online sobre emulação no ns-2, `http://www.isi.edu/nsnam/ns/ns-emulation.html`, acessado em Novembro de 2003.

[13] NS-2, Site do Projeto, `http://www.isi.edu/nsnam/ns/`, acessado em Novembro de 2003.

[14] SIMMCAST, Site do Projeto, `http://www.inf.unisinos.br/~simmcast`, acessado em Novembro de 2003.

[15] B. WHITE, J. LEPREAU, S. GURUPRASAD, "Lowering the Barrier to Wireless and Mobile Experimentation", Proceedings of the First Workshop on Hot Topics in Networks (HotNets-I), October 2002.

[16] B. WHITE et alli. "An Integrated Experimental Environment for Distributed Systems and Networks", Proceedings of the 5th Symposium on Operating Systems Design & Implementation, pp. 255-270, December 2002