Uma Ferramenta para Execução de Simulações em Larga Escala

André Detsch, Gerson G. H. Cavalheiro, Luciano P. Gaspary

Programa de Pós-Graduação em Computação Aplicada (PIPCA)
Universidade do Vale do Rio dos Sinos (UNISINOS)

Resumo. *Este artigo apresenta uma ferramenta com funcionalidades direcionadas à execução de um grande volume de simulações, atuando em todas as fases, desde a definição dos experimentos a serem executados até a geração de gráficos a partir dos resultados obtidos. A partir da especificação de um conjunto de experimentos, é possível realizar a sua execução de forma paralela em diversas máquinas de um laboratório, agilizando o processo como um todo. O desenvolvimento e a evolução da ferramenta baseou-se na experiência prática de uso em diversos estudos, mostrando-se bastante útil na automatização de etapas que, em geral, seriam realizadas de forma manual.*

1. Introdução

A execução de simulações computacionais é uma das mais importantes formas de avaliação de novos protocolos e mecanismos de rede. Entretanto, estudos completos envolvendo simulação tipicamente exigem um grande poder computacional, especialmente pelo fato de que, durante o processo de avaliação, inúmeras execuções independentes se fazem necessárias, variando os parâmetros de entrada (por exemplo, número de *hosts* e largura de banda) e as sementes aleatórias (para que uma boa precisão estatística seja obtida).

Este artigo apresenta uma ferramenta que possibilita ao pesquisador uma fácil descrição e paralelização das simulações em um conjunto de estações e, ao mesmo tempo, propicia facilidades de geração automatizada de gráficos, utilizando linguagens de descrição simples e similares entre si. Por empregar a infra-estrutura já instalada de compartilhamento de arquivos via NFS (*Network File System*, [1]), sua implementação e, principalmente, seu uso se tornam extremamente simples, o que incentiva sua utilização mesmo por quem não possui conhecimentos avançados de processamento distribuído.

A Seção 2 apresenta a arquitetura da ferramenta, apresentando, além dos módulos e suas interações, as linguagens utilizadas. A Seção 3 apresenta um exemplo realístico de uso, descrevendo os passos típicos que envolvem a utilização da ferramenta. Por fim, a Seção 4 encerra o artigo com algumas considerações finais.

2. Arquitetura da ferramenta

A ferramenta é composta de três componentes básicos que são executados independentemente:

- *módulo de geração de tarefas*: interpreta o arquivo contendo a descrição dos experimentos e gera as tarefas correspondentes;
- *módulo de execução*: executado em diversas máquinas em paralelo, consome uma-a-uma as tarefas geradas, realizando a sua execução localmente;
- *módulo de geração de gráficos*: a partir das saídas geradas e de um arquivo de descrição, gera os gráficos no formato indicado.

A Figura 1 ilustra a interação entre os elementos da arquitetura, bem como o fluxo relativo ao processo de execução dos experimentos e plotagem dos resultados. Primeiramente (etapa **1** da figura) o usuário define textualmente no arquivo de descrição, através de linguagem específica (definida na Seção 2.1), os experimentos a serem executados. Acionando-se o módulo de geração de tarefas, o arquivo de descrição é interpretado, e as respectivas tarefas são geradas através da criação de arquivos (cujo nome representa o *id* único da tarefa) no diretório `Waiting/Tasks` (**2**). Este diretório (a exemplo dos demais envolvidos no

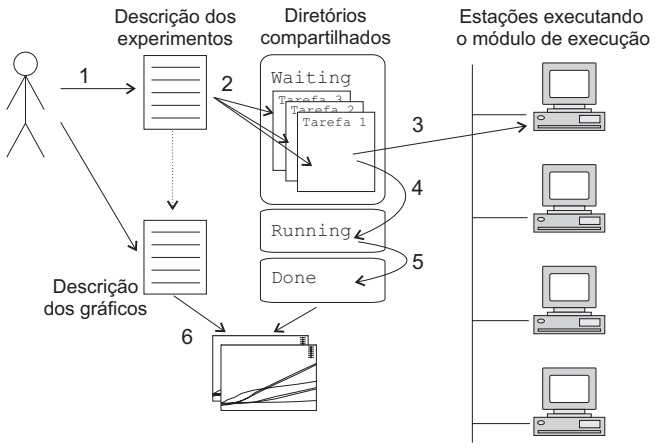

Figura 1: Diagrama de funcionamento da ferramenta

processo aqui descrito) é compartilhado através de NFS entre todas as máquinas que irão contribuir na execução dos experimentos. Nessas máquinas é acionado o módulo de execução, tipicamente com o auxílio de *login* remoto (*Secure Shell* - SSH, por exemplo). As instâncias do módulo monitoram o diretório `Waiting/Tasks` em busca de tarefas para executar (**3**). Assim que uma tarefa apta a ser executada é identificada, ela é transferida para o diretório `Running/Tasks` (**4**) e o comando relativo à tarefa é executado localmente. Nessa etapa, as saídas padrão (*stdout*) e de erro (*stderr*) do comando são redirecionadas para uma arquivo específico (identificado pelo *id* da tarefa) nos diretórios `Running/Output` e `Running/Error`, respectivamente. Quando a execução da tarefa é terminada, o respectivo arquivo é transferido para o diretório `Done/Tasks` (**5**), e os arquivos de saída para os diretórios `Done/Output` e `Done/Error`.

A criação dos gráficos (**6**) utiliza os resultados gerados (no diretório `Done/Output`) e o arquivo de descrição de gráficos. Esse arquivo é criado utilizando a linguagem apresentada na Seção 2.3, que se baseia fortemente na linguagem de descrição de experimentos (Seção 2.1), o que permite reaproveitar boa parte da codificação já realizada.

2.1. Linguagem para definição dos experimentos

O conjunto de simulações a ser realizado pela ferramenta é descrito através de uma linguagem simples, que provê recursos que facilitam a especificação. A seguir são listadas as suas características principais, tendo como base um exemplo fictício, onde é recriada a avaliação de um mecanismo ou aplicação qualquer variando-se o protocolo utilizado e o número de nodos envolvidos.

Definição de um bloco de tarefas

Durante a interpretação do arquivo, tarefas são geradas sempre que um comando `dtask` é encontrado no arquivo de descrição. Isto permite que diversos blocos de tarefas sejam definidos em um mesmo arquivo, de acordo com os diferentes comandos interpretados até a chamada de cada `dtask`. Opcionalmente, pode-se definir um nome para o bloco de tarefas sendo gerado (por exemplo `dtask BLOCO_TCP`). Isto permite a posterior definição de dependência com o bloco de tarefas, conforme ilustrado a seguir, na especificação de requisitos.

Comandos básicos

Os comandos básicos `parameters`, `defaults` e `commandbase` são obrigatórios na especificação de um conjunto de experimentos. `Parameters` define o nome dos parâmetros bem como a ordem na qual serão passados para o simulador, enquanto `defaults` atribui valores padrão para esses parâmetros. Já `commandbase` define o comando de chamada

do simulador. O Código 1 apresenta um exemplo que usa os três comandos. A interpretação do exemplo levaria a geração de apenas uma tarefa cujo comando chamado é "`ns simulacao.tcl TCP 30 0`".

Código 1 Uso dos comandos básicos.

```
1 parameters protocolo nodos semente
2 defaults   TCP       30    0
3 commandbase ns simulacao.tcl
4 dtask
```

Definição da variação dos parâmetros

A definição de diferentes simulações a serem executadas se dá através da variação dos valores dos parâmetros passados para o simulador. Essa variação é especificada através da primitiva `values`, passando-se o nome do parâmetro em questão e os valores desejados, conforme exemplificado nas linhas 4 e 5 do Código 2.

Código 2 Exemplo de variação de parâmetros.

```
1 parameters  protocolo nodos semente
2 defaults    TCP       30    0
3 commandbase ns simulacao.tcl
4 values protocolo TCP UDP
5 values nodos    10 20
6 dtask
```

Com a adição dessas duas linhas, passam a ser geradas não apenas uma tarefa, mas o resultado do produto cartesiano das variações definidas, no caso, quatro tarefas: "`ns simulacao.tcl TCP 10 0`", "`ns simulacao.tcl TCP 20 0`", "`ns simulacao.tcl UDP 10 0`" e "`ns simulacao.tcl UDP 20 0`". Pode-se ainda utilizar comandos Python ([2]) que retornem uma lista de valores no lugar de escrevê-los um a um. Por exemplo, é possível variar o parâmetro "semente" utilizando o comando `values semente range(10)`, de forma que o valor de semente será variado dentre todos os valores inteiros no intervalo [0, 9].

Delimitação de escopo

Um recurso importante para facilitar a especificação de uma grande variedade de experimentos é a delimitação de escopos, utilizando os caracteres { e }. Sempre que um escopo é encerrado, todas as especificações realizadas dentro desse escopo são descartadas. Isto se mostra bastante útil quando conjuntos de experimentos diferentes compartilham apenas algumas especificações (por exemplo, os parâmetros e algumas variações), mas não outras. O Código 3 ilustra essa funcionalidade. A interpretação do código gera quatro tarefas: "`ns simulacao.tcl TCP 30 0`", "`ns simulacao.tcl UDP 30 0`", referentes ao primeiro bloco; e "`ns simulacao.tcl TCP 10 0`", "`ns simulacao.tcl TCP 20 0`", referentes ao segundo bloco.

Código 3 Exemplo de definição de escopo

```
1  parameters protocolo nodos semente
2  defaults   TCP       30    0
3  commandbase ns simulacao.tcl
4  {
5    values protocolo TCP UDP
6    dtask
7  }
8  {
9    values nodos 10 20
10   dtask
11 }
```

Definição de macros

Macros são definidas através do comando `macro`. Por exemplo, `macro numeros_nodos 100 200` define uma macro `numeros_nodos` com o valor `100 200`. Macros podem

também conter diversas linhas, bastando delimitá-las com operadores de escopo ({ e }). Macros são acessadas utilizando o caracter **!** antes do identificador da macro. Por exemplo, o comando `values nodos !numero_nodos` seria expandido para `values nodos 100 200`.

Especificação de requisitos
Uma facilidade mais avançada na especificação de simulações é a definição de requisitos para um conjunto de tarefas. Os requisitos são especificados através da primitiva `requires`, seguido de um dos tipos de requisitos:

- **cpu:** freqüência mínima, em *megaherz*, da máquina na qual a tarefa pode ser executada. Exemplo: `requires cpu 1000`.
- **mem:** quantidade mínima de memória RAM exigida (em *megabytes*). Exemplo: `requires mem 256`.
- **ips:** lista de localidades (endereços IP) que podem executar a tarefa. Exemplo: `requires ips 10.16.165.152 10.16.165.153`.
- **tasks:** blocos de tarefas que devem ser executadas antes do bloco atual. Exemplo: `requires tasks BLOCO_TCP`.

2.2. Arquivos de tarefas

Os arquivos de tarefas são gerados automaticamente pelo módulo de geração de tarefas. O formato desses arquivos é bastante simples e tem como obrigatório apenas uma linha que contenha o comando a ser executado, no formato `RUN: <comando>`, por exemplo, `RUN: ns simulação.tcl TCP 20 0`. Adicionalmente, podem haver linhas de especificação de requisitos, como `REQUIRED_CPU: <MHz>` e `REQUIRED_MEM: <MB>`, inseridas (pelo módulo de geração de tarefas) de acordo com os requisitos especificados pelo usuário. Quando presentes, estas linhas são avaliadas pelo módulo de execução antes do processamento da tarefa. Caso algum requisito não seja atendido, a tarefa é mantida no diretório `Waiting/Tasks`, e outra tarefa é procurada.

2.3. Processamento de resultados e geração de gráficos

Aliado à geração e processamento distribuído das simulações, foi desenvolvida uma ferramenta (*front-end* para o Gnuplot, [3]) que, utilizando uma linguagem similar à apresentada na Seção 2.1, permite a fácil geração de gráficos a partir dos resultados obtidos após a execução das simulações. Enquanto na linguagem de definição de experimentos existe um comando `dtask` que, baseado nas variações definidas, gera as tarefas correspondentes, na linguagem de definição de gráficos existe um comando `dplot` que gera um gráfico, onde cada uma das combinações é representada por uma linha no gráfico gerado.

As funcionalidades de definição de macros, delimitação de escopo e variações dos parâmetros (comando `values`) são tratados da mesma forma. O comando `parameters` também possui a mesma funcionalidade, porém a lista definida deve conter não apenas os parâmetros de entrada do simulador mas também os campos de retorno, que contém os resultados. Os comandos `defaults`, `commandbase` e `requires` são ignorados. Por outro lado, existem comandos que são utilizados apenas na definição dos gráficos. São eles:

- `datafile`: define um ou mais arquivos que contém os dados de entrada para geração dos gráficos;
- `project`: define os dois parâmetros projetados nos eixos X e Y do gráfico. Tanto em X quanto em Y podem ser usadas expressões que definam o eixo, conforme exemplificado na linha 5 do código 4;
- `projectiontype`: especifica a semântica de agrupamento dos valores no eixo Y. O tipo padrão é `mean`, onde é plotada a média dos valores do parâmetro em Y para cada valor de X. Os tipos `max`, `min` e `count` projetam, respectivamente, o valor

máximo, mínimo e o número de ocorrências do valor em X. Já o tipo `confidence` plota, além de uma linha sobre a média dos valores, o intervalo de confiança para 95% de significância.

Adicionalmente, podem ser usados comandos Gnuplot diretamente, permitindo uma total liberdade na formatação dos gráficos, conforme exemplificado na linha 7 do Código 4.

Código 4 Exemplo de descrição de gráfico

```
1  parameters protocolo nodos semente bytes_enviados tempo
2  values protocolo TCP UDP
3  values nodos    10 20
4  datafile resultados1.txt resultados2.txt
5  project nodos bytes_transmitidos / tempo
6  projectiontype confidence
7  set title "Variação da vazão em função do número de nodos"
8  dplot grafico.eps
```

3. Utilização da ferramenta

Esta seção apresenta um exemplo ilustrativo do uso da ferramenta, utilizando o simulador ns-2. O exemplo é baseado nos experimentos realizados em [4], onde cinco modelos básicos de protocolos baseados em *polling* (PET, PeP, PSEW, RBP, PrP) foram avaliados, buscando-se observar e comparar o comportamento desses modelos com a variação, entre outros parâmetros, do número de receptores e tamanho da janela deslizante empregada. O produto final dos experimentos será apresentado em três gráficos: vazão em função do tamanho da janela, custo de rede e vazão em função do número de receptores. Cada um dos gráficos contém cinco linhas, uma para cada protocolo testado.

3.1. Preparação dos experimentos

O primeiro passo consiste no preparo do arquivo de descrição de experimentos. Para o estudo exemplificado nesta seção, o arquivo respectivo é apresentado no Código 5. De forma consistente com esse arquivo, o simulador deve receber os parâmetros na ordem indicada. No caso do ns-2, onde é utilizado um script tcl para instanciação dos experimentos, são utilizadas linhas do tipo `set protocolo [lindex $argv 0]`, uma para cada parâmetro.

Código 5 Arquivo de descrição de experimentos

```
parameters protocolo receptores janela semente
defaults   PET       100        64     0
commandbase ns poll.tcl
values protocolo PET PSEW RBP PeP PrP
values semente range(0,10) # 10 valores, de 0 a 9
# avaliação do impacto da variação do tamanho da janela na vazão
{
  values janela 50 100 150 200 250 300 350 400 450 500
  dtask
}
# avaliação do impacto do número de receptores no custo e vazão
{
  {
    values receptores 1 50 100
    dtask
  }
  {
    values receptores 200 300 400
    requires mem 512 dtask
   dtask
  }
}
```

3.2. Instanciação dos experimentos

Uma vez preparado o arquivo de descrição, as respectivas tarefas podem ser geradas acionando-se o *módulo de geração de tarefas*, através do comando `dtask gen`

<arquivo_de_descrição>. Este comando pode ser executado em qualquer máquina com acesso aos diretórios de tarefas compartilhados. Para que a execução dos experimentos tenha início, deve-se ativar instâncias do *módulo de execução* em diferentes máquinas, através do comando `dtask server`.

3.3. Geração dos gráficos

O Código 6 apresenta o arquivo de descrição referente à geração dos gráficos do exemplo aqui utilizado. O arquivo de dados definido, `resultados.txt`, é construído concatenando-se os resultados escritos nos arquivos em `Done/Output`, tipicamente através de um comando `cat Done/Output/* > resultados.txt`. Vale ressaltar que o formato de saída dos resultados por parte do simulador deve ser consistente com esse arquivo, ou seja, as linhas exibidas devem conter os campos indicados no comando `parameters` do arquivo de descrição de gráficos. O acionamento do *módulo de geração de gráficos* é realizado utilizando-se o comando `dplot <arquivo_de_descrição>`, que pode ser executado em qualquer máquina com acesso ao arquivo de dados indicado.

Código 6 Arquivo de descrição dos gráficos

```
parameters  protocolo receptores janela semente custo tempo
datafile resultados.txt
values protocolo PET PSEW RBP PeP PrP
{
  # um ponto por protocolo para cada tamanho de janela
  project janela custo/tempo     # janela  X  custo/tempo
  set xlabel "Tamanho da Janela" # comando gnuplot
  set ylabel "Vazão (Kbits/s)"   # comando gnuplot
  dplot janela_vazao.eps         # gera o gráfico
}
{
  set xlabel "Numero de Receptores"
  {
    project receptores custo
    set ylabel "Custo de Rede (Mbits)"
    dplot receptores_custo.eps
  }
  {
    project receptores custo/tempo
    set ylabel "Vazão (Kbits/s)"
    dplot receptores_vazao.eps
  }
}
```

4. Considerações finais

Este artigo apresentou uma ferramenta moldada em função das necessidades específicas de estudos que envolvem simulação, em especial, na área de redes de computadores. Desenvolvida em paralelo com diversos projetos de pesquisa que fizeram uso de suas facilidades, o conjunto de módulos que a compõe se mostra bem adaptado aos diferentes requisitos de diferentes estudos. A ferramenta, bem como sua documentação e exemplos mais avançados, podem ser acessados através do site `http://www.gobolinux.org/~detsch/dtools`.

Referências

[1] Network File System Protocol Specification, RFC 1094, http://www.ietf.org/rfc/rfc1094.txt

[2] Python Programming Language, http://www.python.org.

[3] Gnuplot, http://www.gnuplot.info.

[4] M. Barcellos, A. Detsch, "Avaliação de Desempenho de Protocolos para Multicast com Conhecimento de Grupo baseados em Polling". *In: SBRC2002 - XX Simpósio Brasileiro de Redes de Computadores*, 2002, Búzios.