Encontro Nacional do Colégio de Engenharia Electrotécnica 1997
Ordem dos Engenheiros - Colégio de Eng. Electrotécnica

Título:

SIMULADOR DE ESTRATÉGIAS DE DISTRIBUIÇÃO DE CARGA

Autores:

Manuel Alcino Cunha, Luis Paulo Santos, Orlando Belo
Departamento de Informática
Escola de Engenharia
Universidade do Minho
Largo do Paço, 4709 Braga Codex, PORTUGAL
tel: 351-53-604470
fax: 351-53-604471
emails: {alcino,psantos,obelo}@di.uminho.pt

Sumário:

A análise quantitativa de um novo mecanismo de distribuição de carga é um processo fundamental para se verificar se este atingiu os objectivos inicialmente propostos. Esta análise deve ser feita antes da implementação definitiva do mesmo, evitando-se o desperdício de recursos humanos e materiais que poderiam advir de uma análise incompleta ou errada. Um dos métodos mais utilizados para fazer esta análise é a simulação numérica devido à grande flexibilidade que apresenta, quer na definição do sistema de processamento subjacente, quer na definição da carga a que é sujeito. Neste artigo é apresentado um simulador genérico de distribuição de carga desenvolvido em Prolog. Esta linguagem foi escolhida devido à sua reconhecida capacidade para a definição de comportamentos algorítmicos complexos, necessária para modelar muitos dos mecanismos de distribuição actuais.

SIMULADOR DE ESTRATÉGIAS DE DISTRIBUIÇÃO DE CARGA

Alcino Cunha   Luis Santos   Orlando Belo
Dep. Informática, Escola de Engenharia, Universidade do Minho, 4710 Braga, Portugal
Email: {alcino,psantos,obelo}@di.uminho.pt

Sumário

A análise quantitativa de um novo mecanismo de distribuição de carga é um processo fundamental para se verificar se este atingiu os objectivos inicialmente propostos. Esta análise deve ser feita antes da implementação definitiva do mesmo, evitando-se o desperdício de recursos humanos e materiais que poderiam advir de uma análise incompleta ou errada. Um dos métodos mais utilizados para fazer esta análise é a simulação numérica devido à grande flexibilidade que apresenta, quer na definição do sistema de processamento subjacente, quer na definição da carga a que é sujeito. Neste artigo é apresentado um simulador genérico de distribuição de carga desenvolvido em Prolog. Esta linguagem foi escolhida devido à sua reconhecida capacidade para a definição de comportamentos algorítmicos complexos, necessária para modelar muitos dos mecanismos de distribuição actuais.

1 INTRODUÇÃO

Existem dois grandes métodos para se efectuar a análise quantitativa à priori de um mecanismo de distribuição de carga:

- **O método analítico**, onde as estratégias de distribuição são modeladas através de fórmulas analíticas, assumindo-se a possibilidade de quantificar todas as variáveis fundamentais do sistema e também o conhecimento das expressões que definem a dependência relativa dessas variáveis [9]. A maior parte dos modelos utilizados baseiam-se na utilização de redes de filas para representar o sistema de processamento e as tarefas em execução. Consultar [3, 10] como exemplos da utilização deste método.
- **O método da simulação numérica**, onde não se assume o conhecimento detelhado das equações que relacionam as variáveis do sistema, assumindo-se apenas o conhecimento do funcionamento interno dos seus diversos componentes, ou seja, do mecanismo de distribuição. Os resultados são implicitamente obtidos como consequência da interação dos diversos componentes do sistema reproduzidos computacionalmente. Ver, como exemplos, [8, 5, 4, 11].

Existem autores que recorrem à construção de protótipos dos mecanismos desenvolvidos, procedendo depois à análise do seu desempenho num sistema físico real. O sistema é sujeito a uma carga artificial, ou sintética, analisando-se depois o seu comportamento quando em presença de diversos mecanismos de distribuição. O protótipo não precisa de incluir todas as funcionalidades necessárias num mecanismo de distribuição, nomeadamente as que garantem a robustez do mecanismo. No entanto, necessita que as funcionalidades essenciais de distribuição estejam já implementadas, não sendo, por isso, um verdadeiro método de análise à priori. Como exemplos de aplicação deste método ver, por exemplo, [6, 7].

Cada um destes três métodos apresenta, simultaneamente, vantagens e desvantagens, nomeadamente:

- devido à dificuldade em encontrar as expressões que modelam a relação entre as diversas variáveis do sistema, o método analítico apenas pode ser aplicado na análise de estratégias de balanceamento muito simples e nunca em estratégias caracterizadas por um comportamento complexo;
- a grande vantagem do método analítico consiste na rapidez de obtenção de resultados, mesmo quando a solução das expressões encontradas tem de ser obtida através de métodos numéricos

iterativos;

- a simulação com carga artificial implica um grande esforço no desenvolvimento dos protótipos dos mecanismos em análise, pois estes têm de operar sobre um sistema real;
- este método também implica que o sistema de processamento seja dedicado exclusivamente ao processo de análise, situação fácil de se conseguir quando se trata de uma máquina paralela, mas muito difícil em sistemas distribuídos devido à grande quantidade de utilizadores e devido ao grande número de tarefas do sistema operativo em execução;
- quer o método analítico, quer o método de simulação com carga artificial, são muito rígidos quanto possibilidade de analisar o mecanismo sobre diferentes configurações físicas, devido, respectivamente, à complexidade de modelar as expressões inerentes a configurações mais complexas, e à dificuldade ou impossibilidade de alterar fisicamente a configuração do sistema de processamento subjacente;
- o método da simulação numérica requer a concepção de um simulador que suporte os diversos mecanismos de distribuição em análise, sendo, no entanto, esta tarefa mais simples do que a construção dos protótipos para a simulação com carga artificial;
- este método é o mais flexível quanto à capacidade de analisar diferentes configurações do sistema subjacente, diferentes padrões de carga nesse sistema e diferentes configurações dos elementos de distribuição;
- o tempo requirido pela simulação numérica pode ser relativamente elevado se o número de entidades envolvidas na simulação for muito grande.

Devido à flexibilidade que apresenta ao nível da configuração do sistema subjacente, e devido à possibilidade de se definir diferentes padrões de carga e diferentes configurações dos elementos de distribuição, achamos que o método da simulação numérica é o mais adequado para analisar à priori um novo mecanismo de distribuição de carga. Formalmente, o modelo de simulação que vai ser usado caracteriza-se por ser [9]: abstracto, numérico, dinâmico, explícito e estocástico.

2 PROCEDIMENTO DA SIMULAÇÃO

Por questões de flexibilidade, e devido ao facto da simulação poder ser um processo extremamente demorado, dividiu-se o processo em duas fases distintas (figura 1), nomeadamente:

- a simulação propriamente dita, onde a partir de uma determinada configuração, armazenada num ficheiro, se geram os resultados da mesma, sendo estes também armazenados em ficheiro; e
- a análise de resultados, feita mais tarde recorrendo aos ficheiros de resultados gerados pelo simulador.

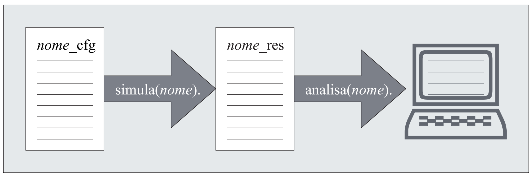

Figura 1: Pocedimento da Simulação

Uma vantagem desta separação é permitir que o processo de simulação seja feito em *batch*. Outra vantagem consiste em não ter de se calcular todas as estatísticas durante o processo de simulação, o que o tornaria ainda mais lento e obrigaria à definição à priori de todas as estatísticas necessárias. Se assim não fosse, de todas as vezes que se pretende-se obter uma nova estatística seria necessário executar nova simulação.

Todas as rotinas necessárias no processo de simulação, nomeadamente o próprio simulador e as rotinas para análise estatística dos resultados, foram desenvolvidas no SICStus [2] Prolog. Esta linguagem não é particularmente rápida, especialmente na versão interpretada utilizada, não sendo por isso, à partida, uma grande candidata à construção de um simulador cujo grande problema é o tempo de execução. No entanto, esta linguagem apresenta grandes vantagens, nomeadamente:

- capacidade de modelar comportamentos algorítmicos complexos e grandes capacidades de inferência, necessárias para modelar a complexidade crescente dos mecanismos de distribuição de carga e modelar fielmente as características do sistema de processamento subjacente;
- pequenos tempos de desenvolvimento para permitir modelar um maior número de mecanismos de distribuição, tornando a análise mais efectiva.

A primeira etapa para se efectuar uma simulação consiste na elaboração do ficheiro de configuração do sistema em análise, descrito na secção 3. Este ficheiro poderá mais tarde ser analisado estatisticamente recorrendo a uma biblioteca de predicados de análise. O funcionamento interno do simulador é descrito na secção 4.

3 FICHEIRO DE CONFIGURAÇÃO

O ficheiro de configuração é uma das partes fundamentais do processo de simulação, porque expressa toda a flexibilidade do simulador e porque constitui a interface com o utilizador final. A sintaxe deste ficheiro é apresentada na figura 2 em notação BNF (os termos delimitados por "/ /"representam termos não terminais cuja expansão é trivial e não relevante para a especificação). Na figura 3 é apresentada a configuração correspondente a uma máquina paralela com 4 nodos e uma topologia em matriz e com agentes de distribuição de carga com uma estratégia aleatória.

Para facilitar o processo de simulação, o simulador procede a uma análise sintática e semântica do ficheiro de configuração, reportando os erros respectivos ao utilizador.

A seguir apresentam-se mais detalhadamente os diversos parâmetros que caracterizam as entidades presentes no ficheiro de configuração, explicando-se em que medida é que influenciam o seu comportamento durante a simulação. Não é apresentada qualquer explicação em relação à entidade organização (grupo), pois esta é uma entidade perfeitamente passiva, correspondendo apenas a uma relação de dados.

3.1 Controlo da Simulação

Nesta secção apresentam-se os parâmetros dos predicados de controlo da simulação, comuns a todas as entidades da mesma, nomeadamente:

- **Semente.** Ajusta a semente do gerador de números aleatórios utilizado. É essencial para permitir efectuar diferentes simulações com a mesma configuração, aumentando assim o nível de confiança nos resultados da mesma.

- **Inicio de monitorização.** Utilizado pelos predicados de análise estatística dos resultados. Indica a partir de que instante serão consideradas as tarefas chegadas ao sistema, servindo para ignorar o período de inicialização em que o sistema ainda não atingiu um estado estacionário [9], nomeadamente no respeitante ao número de tarefas no sistema.

- **Fim de simulação.** Indica quando termina a simulação.

- **Imprime tempo.** Indica de quanto em quanto tempo será comunicado o tempo actual de simulação, e de quanto em quanto tempo será feito o *garbage collection* das *stacks* do Prolog.

- **Tamanho das mensagens de controlo.** Indica o tamanho (em bytes) das mensagens de controlo trocadas entre as entidades de simulação, como por exemplo, os pedidos de distribuição de tarefas ou as sondagens. Deve ser ponderado com cuidado para não desvirtuar os resultados finais.

- **Cabeçalho.** Indica o tamanho do cabeçalho de uma mensagem. Constitui o limite inferior no tamanho de uma mensagem que circula na rede. Depende do protocolo de comunicação que se pretenda simular.

```
% Controlo da simulacao
semente(27134, 9213, 17773).
inicio_monitorizacao(1000).
fim_simulacao(20000).
imprime_tempo(1000).
tamanho_mensagens_controle(100).
cabecalho(50).
% Grupos
grupo(grupo1, "Organizacao XXX").
% Nodos
nodo(nodo1, grupo1, [linha1, linha2], [(cpu, 1000, activo),
(memoria, 20000, passivo)], [(0, 20000, carga1)], (100, 50)).
nodo(nodo2, grupo1, [linha1, linha3], [(cpu, 1000, activo),
(memoria, 20000, passivo)], [(0, 20000, carga1)], (100, 50)).
nodo(nodo3, grupo1, [linha2, linha4], [(cpu, 1000, activo),
(memoria, 20000, passivo)], [(0, 20000, carga1)], (100, 50)).
nodo(nodo4, grupo1, [linha3, linha4], [(cpu, 1000, activo),
(memoria, 20000, passivo)], [(0, 20000, carga1)], (100, 50)).
% Linhas
linha(linha1, nodo1, 20000).
linha(linha2, nodo1, 20000).
linha(linha3, nodo1, 20000).
linha(linha4, nodo1, 20000).
% Tipos de carga
carga(carga1, expneg(1000), [(50, tarefa1), (50, tarefa2)]).
% Tipos de tarefas
tarefa(tarefa1, expneg(700), nao_tem, [(cpu, exactamente(1000),
activo)], (nao_tem, nao_tem), exactamente(100)).
tarefa(tarefa2, expneg(700), nao_tem, [(cpu, exactamente(1000),
activo)],(exactamente(100),exactamente(100)),exactamente(100)).
% Agentes de Distribuição Aleatórios
aleatorio(aleatorio1, nodo1, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio2, nodo2, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio3, nodo3, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio4, nodo4, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio5, nodo5, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio6, nodo6, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio7, nodo7, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio8, nodo8, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
aleatorio(aleatorio9, nodo9, 1, 1,
          [nodo1, nodo2, nodo3, nodo4]).
```

Figura 3: Exemplo de ficheiro de configuração

```
<configuracao> ::= {<comentario>} <controlo> {<entidade>}

<comentario> ::= % /string/

<controlo> ::= <semente> <inicio> <fim> <monitorizacao> <mensagens_controle> <cabecalho>
<semente> ::= semente(/inteiro/, /inteiro/, /inteiro/).
<inicio> ::= inicio_simulacao(/inteiro/).
<fim> ::= fim_simulacao(/inteiro/).
<monitorizacao> ::= imprime_tempo(/inteiro/).
<mensagens_controle> ::= tamanho_mensagens_controle(/inteiro/).
<cabecalho> ::= cabecalho(/inteiro/).

<entidade> ::= <comentario> | <grupo> | <nodo> | <linha> | <carga> | <tarefa> | <agente>

<grupo> ::= grupo(/identificador/, "/string/").

<nodo> ::= nodo(/identificador/, /identificador/, <lista_identificadores>,
           <recursos_nodo>, (/inteiro/, /inteiro/)).
<lista_identificadores> ::= [] | [{/identificador/,} /identificador/]
<recursos_nodo> ::= [{<recurso_nodo>,} <recurso_nodo>]
<recurso_nodo> ::= (/identificador/, /inteiro/, <tipo>)
<tipo> ::= activo | passivo

<linha> ::= linha(/identificador/, /identificador/, /inteiro/).

<carga> ::= carga(/identificador/, <distribuicao1>, <lista_tarefas>).
<distribuicao1> ::= exactamente(/inteiro/) | expneg(/inteiro/)
<lista_tarefas> ::= [{<tarefa_carga>,} <tarefa_carga>]
<tarefa_carga> ::= (/inteiro/, /identificador/)

<tarefa> ::= tarefa(/identificador/, <distribuicao1>, <distribuicao2>, <recursos_tarefa>,
                   (<distribuicao2>, <distribuicao2>), <distribuicao1>).
<distribuicao2> ::= <distribuicao1> | nao_tem
<recursos_tarefa> ::= [{<recurso_tarefa>,} <recurso_tarefa>]
<recurso_tarefa> ::= (/identificador/, <distribuicao1>, <tipo>)

<agente> ::= <agente_aleatorio> | <faltam os outros>

<agente_aleatorio> ::= aleatorio(/identificador/, /identificador/, /inteiro/,
                       /inteiro/, <lista_identificadores_nao_vazia>).
<lista_identificadores_nao_vazia> ::= [{/identificador/,} /identificador/]
```

Figura 2: Sintaxe de um ficheiro de configuração

3.2 Nodos

Os parâmetros que caracterizam um nodo são os seguintes:

- **Identificação.** Caracteriza univocamente cada nodo do sistema.

- **Organização a que pertence.** Serve para modelar um sistema multi-organizacional.

- **Linhas a que está ligado.** Define a topologia da rede de processamento e indica, caso o nodo esteja ligado a mais do que uma linha, se o nodo vai ser sujeito a sobrecarga devido ao encaminhamento de mensagens para outros nodos.

- **Recursos.** Define a heterogeneidade do sistema ao nível do sistema de processamento. Cada recurso caracteriza-se pela sua identificação (única dentro de cada nodo), pela capacidade respectiva e pelo tipo de recurso. Quanto ao tipo, os recursos podem ser activos ou passivos. Os recursos activos são partilhados pelas tarefas em execução no nodo (*time-sharing*). É possivel definir prioridades no escalonamento reccorrendo à reserva de uma percentagem de um recurso para uma determinada tarefa. Quando não existem reservas, os recursos activos são equitativamente distribuídos pelas tarefas em execução variando a percentagem atribuída com o número de tarefas. Os recursos passivos também podem ser partilhados (como, por exemplo, um ficheiro de leitura), sendo essa situação caracterizada pelo valor -1 presente no atributo capacidade. A capacidade de um recurso passivo requirida por uma determinada tarefa é-lhe alocada durante o tempo total da sua execução.

- **Padrão de carga.** Serve para modelar o padrão de carga a que o nodo será sujeito. Permite a definição de um padrão variável ao longo do tempo, sendo cada elemento da lista constituído por um intervalo de tempo (instante inicial mais instante final) e pela identificação do padrão respectivo (ver definição de padrões de carga na secção 3.4). Esta possibilidade é importante para modelar a realidade dos sistemas distribuídos, caracterizados por períodos com carga muito baixa (nocturnos) seguidos por períodos muito sobrecarregados (horas de expediente). A possibilidade de modelar diferentes padrões para cada nodo é necessária, dado, por exemplo, o desfazamento que existe entre os picos de carga em sistemas afastados geográficamente ou entre nodos de um mesmo sistema. Define a heterogeneidade do sistema ao nível da carga.

- **Tempos de processamento de mensagens.** Indicam os tempos de CPU necessários para fazer, respectivamente, o *packing/unpacking* de mensagens originárias ou destinadas ao nodo e para fazer o encaminhamento de mensagens destinadas a outros nodos (relevante apenas quando o nodo está ligado a mais do que uma linha). Os valores indicados são factores que serão multiplicados pelo tamanho das mensagens, sendo o total obtido o tempo de processamento requirido para um processador com capacidade unitária (para facilitar a comparação entre os tempos dos diversos nodos). Estes parâmetros são específicos para cada nodo, de modo a reforçar a possibilidade de definição de sistemas heterogéneos.

3.3 Linhas de Comunicação

As linhas de comunicação são caracterizadas pelos seguintes parâmetros:

- **Identificação.** Caracteriza univocamente cada linha de comunicação.
- **Nodo responsável.** Define qual o nodo que controla o acesso e taxação na linha. Implicitamente também define qual a organização sua proprietária.
- **Capacidade.** Indica qual a velocidade da linha expressa em bytes por 1000 unidades de simulação. Se considerarmos que cada unidade de simulação corresponde a um milisegundo, então teremos bytes por segundo.

Convêm apresentar qual o processo de encaminhamento de mensagens utilizado no simulador. No início da simulação cada nodo e cada linha do sistema calcula as suas tabelas de encaminhamento. Estas tabelas indicam qual a linha (no caso dos nodos) ou qual o nodo (no caso das linhas) que uma mensagem deve seguir para atingir um outro nodo do sistema. Presentemente, essas tabelas são construídas tendo por base o caminho mais rápido (teoricamente) entre a entidade que constroi a tabela e os restantes nodos do sistema. Este processo não garante o funcionamento mais eficiente em todos os casos e deverá ser alvo de futuros melhoramentos.

O predicado disponibilizado para o envio de mensagens permite o *multi-cast*, ou seja, permite definir o destino da mensagem como um qualquer subconjunto de todas entidades existentes no sistema. Se apenas indicarmos uma entidade teremos comunicação ponto a ponto e se indicarmos todas as entidades teremos comunicação por difusão. Quando uma linha, ou nodo, tem de entregar uma mensagem as entidades destino são agrupadas de acordo com a próxima entidade por onde deverão ser encaminhadas. A mensagem original é depois partida em tantas mensagens quantas as próximas entidades determinadas, sendo essas mensagens encaminhadas para essas entidades, onde recomeça este processo. Com este procedimento garante-se que será gerado o número mínimo de mensagens possível para entregar a mensagem aos destinos.

3.4 Carga do Sistema

A definição da carga do sistema divide-se em duas partes: a definição de padrões de carga e a definição das tarefas que são utilizadas nesses padrões. Um padrão de carga caracteriza-se pelos seguintes parâmetros:

- **Identificação.** Caracteriza univocamente cada padrão de carga.
- **Tempo entre chegadas de tarefas.** Indica qual a distribuição teórica que modela o tempo médio de intervalo entre chegadas de tarefas a um nodo com este padrão de carga. Quando se pretender que o tempo entre chegadas seja determinístico utiliza-se o predicado `exactamente(`*`Valor`*`)`.
- **Histograma de tarefas.** Define, recorrendo a um histograma, que tipo de tarefas serão geradas. Para cada elemento da lista indica-se qual a percentagem de ocorrências desse tipo de tarefa.

Cada tipo de tarefa caracteriza-se pelos seguintes parâmetros:

- **Identificação.** Identifica univocamente cada tipo de tarefa.
- **Duração.** Indica qual a duração da tarefa (através de uma distribuição teórica) caso fosse executada num nodo de processamento com capacidades de recursos activos idênticas às apresentadas na lista de recursos da tarefa. O cálculo do tempo real de execução num qualquer nodo processa-se da seguinte forma: calcula-se qual o menor ratio entre os recursos activos atribuídos à tarefa (quer seja resultante da partilha, quer seja reservado) e os respectivos recursos activos presentes na lista de recursos da tarefa. Depois, esse factor é multiplicado pelo tempo de execução da tarefa, determinando-se assim o seu tempo real de execução. Este tempo é sistematicamente reavaliado conforme entram ou saiem tarefas da lista de execução do nodo.
- **Prazo de execução.** Indica qual o prazo que a tarefa tem para ser executada, contado a partir do instante de chegada da tarefa ao sistema. A não existência de qualquer prazo é indicada pelo parâmetro `nao_tem`. Caso exista prazo o agente de distribuição assume o conhecimento dos parâmetros de execução da tarefa.
- **Lista de recursos.** Indica quais os recursos que um nodo deve possuir para que a tarefa possa ser executada. As capacidades nos recursos activos não indicam a capacidade que deve ser alocada à tarefa, servindo apenas para determinar o tempo real de execução, tal como foi explicado anteriormente. A capacidade indicada nos recursos passivos deverá existir no nodo e será alocada à tarefa durante o tempo total da sua execução.
- **Necessidades de comunicação.** Define, respectivamente, o intervalo entre envios consecutivos de mensagens para o nodo originário da tarefa e o tamanho dessas mensagens. Quando é enviada uma mensagem deste tipo, a tarefa é retirada de execução, sendo escalonada de novo aquando da chegada de uma mensagem de resposta enviada pelo seu nodo originário. Os únicos atrasos induzidos no nodo originário consistem nos tempos de *unpacking* e *packing* da mensagem enviada pela tarefa e da mensagem de resposta, respectivamente.
- **Tamanho da tarefa.** Indica o tamanho da tarefa (código e dados) em bytes. É usado aquando das transferências de tarefas entre nodos para determinar o tamanho das mensagens.

4 FUNCIONAMENTO DO SIMULADOR

O simulador desenvolvido segue uma filosofia *event-schedulling* [9], ou seja, quando ocorre um evento, a entidade que o gerou deve proceder ao planeamento de todos os eventos futuros que são consequência desse evento. Nesta simulação o evento principal, que desencadeia todo o processo, é a chega de uma tarefa ao sistema. O planeamento das primeiras chegadas é feito no instante zero de simulação. Depois, durante o processo de tratamento de uma nova chegada, cada nodo planeia a próxima. Este processo de planeamentos futuros repete-se para todos os outros eventos.

Todas as entidades activas na simulação, ou seja, os nodos, as linhas de comunicação e os agentes de distribuição, foram desenvolvidas recorrendo a linguagem de padrões [1]. Esta linguagem é ideal para descrever comportamentos reactivos, tal como os apresentados por estas entidades, que reagem ao surgimento de eventos a elas destinados. O interpretador de padrões apresentado em [1] teve de ser ligeiramente alterado para permitir a convivência de padrões referentes a diferentes entidades.

```
%
% Simulador
%

% Inicio simulacao
simulador ::
        [not(tempo(_))]
        --->
        [semente(X, Y, Z),
        setrand(rand(X, Y, Z)),
        assert(tempo(0)),
        assert(tarefa(0)),
        assert(tarefa_aux(-1)),
        todas_entidades(Todos),
        assert(evento(0, Todos, inicia_simulacao)),
        fim_simulacao(Fim),
        assert(evento(Fim, Todos, fim_simulacao)),
        imprime_tempo(Imp),
        assert(evento(Imp, simulador, imprime_tempo)),
        format('Inicio da simulacao.~n', [])].

% Monitorizacao
simulador ::
        [tempo(T),
        evento(T, simulador, imprime_tempo)]
        --->
        [retract(evento(T, simulador, imprime_tempo)),
        format('Tempo = ~w~n',[T]),
        imprime_tempo(Imp),
        NovoT is T + Imp,
        garbage_collect,
        assert(evento(NovoT, simulador, imprime_tempo))].

% Fim de simulacao
simulador ::
        [tempo(T),
        evento(T, [], fim_simulacao)]
        --->
        [retractall(evento(_, _, _)),
        retractall(tempo(_)),
        format('Fim da simulacao.~n', []),
        stop].

% Despoletar eventos nos agentes
simulador ::
        [tempo(T),
        evento(T, [(_, Tipo_agente)|_], _)]
        --->
        [run(Tipo_agente)].

% Limpeza de mensagens vazias
simulador ::
        [evento(T, [], Tipo)]
        --->
        [retract(evento(T, [], Tipo))].

% Avanco do tempo
simulador ::
        [tempo(T),
        not(evento(T, _, _))]
        --->
        [findall(X, evento(X, _, _), Tempos),
        min_list(Tempos, NovoT),
        replace(tempo(T), tempo(NovoT))].
```

Figura 4: *Kernel* do simulador

O próprio *kernel* do simulador foi desenvolvido em linguagem de padrões (ver figura 4). Este *kernel* constitui o "motor" da simulação, estando encarregado da gestão dos eventos e do avanço do tempo. Mais concretamente, as suas funções são:

- **Inicializar a simulação.** As seguintes acções são realizadas no início da simulação: é inicializado o ambiente, é planeado o evento que origina o fim da simulação, planeado o primeiro evento de monitorização, e planeado o evento que marca o início de simulação em todas as entidades presentes. Cada entidade, após a detecção deste evento, inicializa o seu ambiente particular e, no caso dos nodos, planeia a primeira chegada de tarefas.

- **Monitorização.** Imprime o tempo actual de simulação e faz o *garbage collection* das pilhas do interpretador de Prolog.

- **Despoletar eventos nos agentes.** Cada evento é caracterizado pela sua descrição, pelo instante em que vai ocorrer e pela lista de entidades que afecta. Quando se atinge o instante em que o evento vai ocorrer, é necessário activar todas as entidades que estão presentes na lista de entidades afectadas.

- **Retirar eventos que não afectam entidades.** Por questões de optimização, todos os eventos que já não vão afectar nenhuma entidade são retirados do ambiente.

- **Avanço do tempo.** Quando já não existe mais nenhum evento planeado para um dado instante o tempo de simulação é avançado para o tempo do próximo evento planeado. Como a filosofia seguida é a de *event-schedulling* garante-se que durante esse intervalo nada acontece.

- **Fim de simulação.** Quando o evento que marca o fim de simulação já foi despoletado em todas as entidades presentes, o próprio *kernel* termina, limpando o ambiente de todos os eventos planeados para depois desse instante.

Para garantir que a duração real da simulação cresça linearmente em relação ao tempo que se pretende simular é necessário garantir que, quer a base do interpretador de Prolog, quer as suas pilhas de controlo, se mantenham com um tamanho sensivelmente constante ao longo da simulação. Quanto ao primeiro aspecto, optou-se por se armazar os resultados da simulação em ficheiro conforme vão sendo obtidos. Para garantir o segundo ponto foi necessário eliminar a possibilidade de *backtracking* do interpretador.

Por razões de espaço não será apresentado o funcionamento detalhado de todos os agentes de distribuição envolvidas na simulação. Apenas serão apresentados dois exemplos de predicados essenciais ao desenvolvimento dos mesmos, nomeadamente:

- `envia_mensagem((`*`Agente`*`, `*`Tipo`*`), `*`ListaDestino`*`, `*`Mensagem`*`, `*`Tam`*`).` Predicado usado para enviar uma mensagem a uma entidade remota. Os parâmetros indicam, respectivamente, a origem da mensagem, a lista de agentes destino (cada elemento é um tuplo formado pela identificação do agente e o seu tipo), a mensagem a enviar, e o tamanho da mesma. As mensagens quando chegam aos nodos de destino são escalonadas da mesma forma que os eventos normais.

- `distribuidor_presente(`*`Distribuidor`*`, `*`Tipo`*`).` Mensagem que um agente de distribuição de carga deve enviar a um nodo para lhe indicar que é encarregado pela distribuição das suas tarefas.

5 CONCLUSÕES E TRABALHO FUTURO

Neste artigo foi apresentada uma plataforma de simulação para problemas de distribuição de carga. Esta plataforma permite especificar um sistema distribuído multi-organizacional com padrões dinâmicos de carga sendo genérica para qualquer estratégia de distribuição. Para além de permitir comparar a eficiência de diversas estratégias de distribuição, constitui uma ferramenta de apoio aos administradores de sistema, pois permite testar à priori os efeitos da introdução de uma nova estratégia.

Para trabalho futuro, pensa-se generalizar o simulador desenvolvido para problemas genéricos de alocação de recursos e não apenas problemas de distribuição de carga. Também será desenvolvida uma ferramenta integrada, preferencialmente visual, que permita uma definição facilitada de configurações de sistemas e de consulta dos resultados. Outros aspectos importante são a obtenção de intervalos de confiança para os resultados de simulação e a determinação automática das durações óptimas da simulação e dos períodos de inicialização

Referências

[1] Ivan Bratko. *PROLOG: Programming for Artificial Intelligence.* International Computer Science Series. Addison-Wesley, second edition, 1990.

[2] M. Carlsson and J. Widen. *SICStus Prolog User's Manual.* Swedish Institute of Computer Science, 1993.

[3] Derek L. Eager, Edward D. Lazowska, and John Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, SE-12(5):662–675, May 1986.

[4] O. Kremien and J. Kramer. Methodical analysis of adaptive load sharing algorithms. *IEEE Transactions on Parallel and Distributed Systems*, 3(6):747–760, November 1992.

[5] O. Kremien, J. Kramer, and J. Magee. Scalable load-sharing for distributed systems. In *HICSS-26*, January 1993.

[6] Thomas Kunz. The influence of different workload descriptions on a heuristic load balancing scheme. *IEEE Transactions on Software Engineering*, 17(7):725–730, July 1991.

[7] R. Lüling, B. Monien, and F. Ramme. A study on dynamic load balancing algorithms. Technical Report TR-001-92, Paderborn Center for Parallel Computing, June 1992.

[8] Krithi Ramamritham, John A. Stankovic, and Wei Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, August 1989.

[9] A. Guimarães Rodrigues. *Simulação.* Universidade do Minho, 1988.

[10] Marvin M. Theimer and Keith A. Lantz. Finding idle machines in a workstation-based distributed system. *IEEE Transactions on Software Engineering*, 15(11):1444–1458, November 1989.

[11] Songnian Zhou. A trace-driven simulation study of dynamic load balancing. *IEEE Transactions on Software Engineering*, 14(9):1327–1341, September 1988.