Otimização usando Programação em Lógica com Restrições

Gestão de Servidores *Cloud*

Duarte Duarte - ei11101, Hugo Freixo - ei11086
FEUP-PLOG, Turma 3MIEIC1, Grupo 45

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Resumo O objetivo do trabalho é otimizar servidores *cloud*, para a realização deste foram utilizadas restrições na linguagem Prolog. Conclui-se que os módulos de PLR de Prolog não são adequados quando a quantidade de informação a processar é elevada e que uma fraca especificação dos requisitos do problema leva a um trabalho atribulado sem conclusões pertinentes.

1 Introdução

Este trabalho tem como objetivo otimizar a gestão de um servidor *Cloud* através da linguagem Prolog. Pretende-se então simular pedidos a um servidor principal de distribuição (SPD), este apenas pode processar um pedido de cada vez, se for efetuado mais algum pedido enquanto outro está a ser processado no SPD, o pedido fica pendente numa fila de espera. O tamanho da fila é variável mas um novo slot tem um custo extra.

Todos os dados referentes à configuração base dos servidores, tempos de execução de tarefas, preços de hardware e requisitos dos diferentes tipos de pedidos podem ser alterados através de um ficheiro de texto.

O grupo achou o enunciado mal formulado e longe de uma situação real, por isso o problema foi relaxado de modo a facilitar a implementação dos detalhes complexos do problema e para obter uma solução, embora na mesma para valores baixos de pedidos, em tempo útil.

Este relatório é constituido por vários pontos. Começa por explicar o problema proposto ao grupo, depois aborda os ficheiros de dados utilizados para o *input* dos valores das variáveis do problema. Seguidamente são aborados os tópicos acerca das variáveis de decisão e restrições utilizadas na resolução deste trabalho, será descrita a funçaõ de avaliação, a estratégia de pesquisa e a visualização da solução. Este relatório finaliza com os resultados obtidos e com a conclusão e prespectivas de desenvolvimento.

2 Descrição do Problema

Para além do SPD referido no tópico acima, existem mais três tipos de servidores: servidor de autenticação, servidor de armazenamento e servidor de

execução de scripts. Estes servidores vão dar reposta a outros pedidos dependendo das suas exigências. Também existem vários tipos de pedidos e estes possuem um valor de espaço de disco, RAM e CPU que necessitam para serem processados. Estes valores são lhes cedidos pelos servidores e é escolhido o servidor mais apropriado para o tipo de pedido, por exemplo: se queremos guardar um ficheiro então utiliza-se uma máquina que tem muito espaço em disco; se queremos executar um programa então utiliza-se uma maquina que tem um nível de CPU elevado.

Processo seguido:

- Gerar uma lista ordenada aleatoriamente com os tipos de pedidos, respeitando as percentagens de cada tipo e número de pedidos especificados nos dados de entrada;
- Gerar múltiplas listas de *task*s, com tempos de início e fim indefinidos, mas com duração, recurso a gastar e máquina em que corre definido;
- Gerar múltiplas listas de *machine*s com os recursos apropriados (para um total de 9 máquinas (3 cada tipo x 3 cada recurso));
- Definir domínios às variáveis;
- Aplicar restrições;
- Pesquisar pela solução, obtendo a lista de tempos de início e fim de cada pedido;
- Ordenar e selecionar os tempos de início das tarefas;
- Usar o tempo de início das tarefas para calcular o número de "slots"a ser usados;
- Pegar em todos os dados calculados e calcular custo total e apresentar estatísticas e resultados.

3 Ficheiros de Dados

Os ficheiros de dados contêm os dados referentes à configuração base dos servidores (espaço, RAM e CPU dos servidores de autenticação, armazenamento e execução de *scripts*), os tempos de execução de tarefas e decisões de encaminhamento, o preço dos "slots"na fila de pedidos, requisitos dos vários tipos de pedidos, o número de pedidos a simular e a sua distribuição, em percentagem, e o orçamento disponível.

Estes ficheiros de texto são código válido de Prolog, constituídos pela declaração de fatos simples, por exemplo, *spd_slot_price(50).*.

A escalibilidade da solução pode ser verificada apenas alterando o número de pedidos (*requests(X).*). Foi testada um número de pedidos de 100, 1000 e 2500.

4 Variáveis de Decisão

Cada pedido real corresponde a 3 *task/5* distintas, cada uma correspondendo a um tipo de recurso: RAM, CPU ou espaço. Da mesma forma, cada máquina real corresponde a 3 *machine/2*. As *task*s são executadas na máquina respectiva,

tendo em conta o recurso que usam e o seu tipo (autenticação, armazenamento ou *script*). As máquina conseguem executar tarefas paralelamente desde que tenham recursos para tal, caso contrário, a tarefa é adiada.

Como variáveis de decisão temos o tempo de início e fim das tarefas, que tenta ser minimizado. O seu domínio é entre o tempo que demora a processar um pedido no SPD (no exemplo de configuração em anexo é 100 milisegundos) e o número de milisegundos num dia.

5 Restrições

A nível de restrições são usados 3 *cumulatives/3* (um para tipo de recurso) de forma a associar as tarefas às máquinas. Como opção, é usado *bound(upper)* porque os tempos não têm um limite superior bem definido (embora seja objetivo a minimização deles).

6 Função de Avaliação

Depois de calculado os tempos de execução das tarefas ótimos, é calculado o número de slots no SPD necessários (sem recurso a PLR), ver predicado *calculate_cost_queues/4*.

7 Estratégia de Pesquisa

A estratégia de etiquetagem é a mais simples possível, apenas com recurso à chamada *labeling([], StartTimes + End)*.. Não foi possível forçar a minimização dos tempos de terminação, com recurso a *minimize/2* porque não era possível obter soluções em tempo útil.

8 Visualização da Solução

No fim do processamento dos dados de entrada é imprimido para o ecrã um conjunto de dados, tanto estatísticos como de resultados, sobre o problema.

Começa-se por mostrar estatísticas referentes ao módulo de PLR, *fd_statistics/0*, e de seguida estatísticas sobre o tempo de processamento do Prolog e memória usada, *statistics/0*. Quanto aos resultados, é mencionado o número de pedidos de cada tipo, o número de "slots"comprados para o SPD e SSD, o número de servidores comprados de cada tipo, o dinheiro que foi gasto no dimensionamento do sistema e o lucro ou prejuízo relativamento ao orçamento disponível inicialmente.

Exemplo de output:

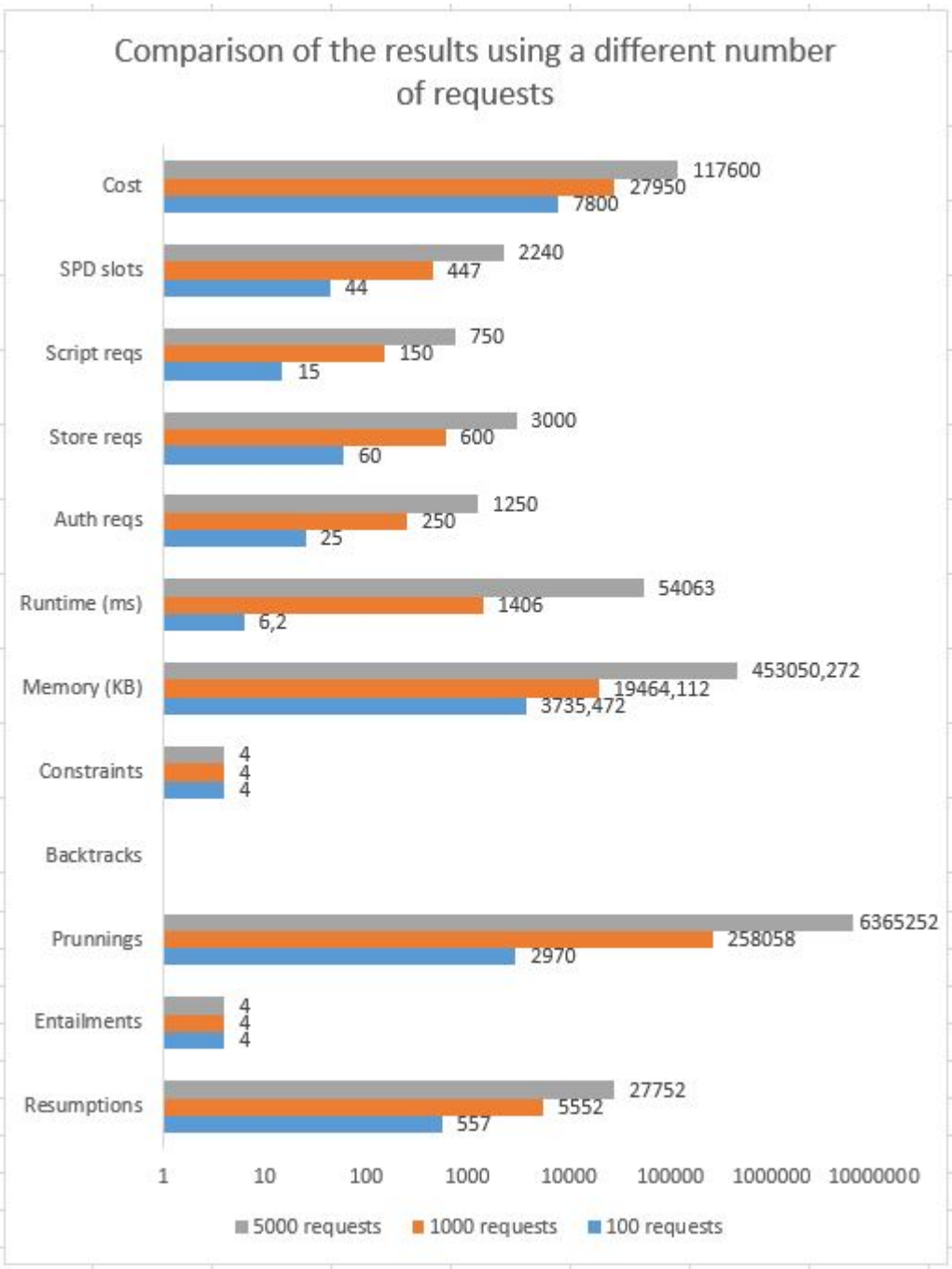
```
| ?- init.
===== FD Statistics =====
Resumptions: 27752
Entailments: 4
```

```
Prunings: 6365252
Backtracks: 0
Constraints created: 4

======= Statistics ======
memory (total)     453050272 bytes
-- snip ---
      54.063 sec. runtime
-- snip ---

======== Results ========
Number of Authentication Requests: 1250
Number of Storage Requests: 3000
Number of Script Execution Requests: 750
Number of SPD slots bought: 2240
Number of SSD slots bought: 0
Number of Authentication Servers: 1
Number of Storage Servers: 1
Number of Script Execution Servers: 1
Available budget: 150000E
Total cost: 117600E
Profit/Loss: +32400E
```

9 Resultados



No gráfico em cima são comparados as diferentes estatísticas e resultados para três diferentes numéro de pedidos: 100, 1000 e 5000. O número de pedidos de teste é relativamente baixo em relação a um sistema de realidade ou ao que era sugerido no enunciado do problema (cem milhões de pedidos) porque aumentando o número de pedidos aumenta exponencialmente (e de forma bastante acentuada) a RAM e tempo necessário à resolução do problema. Nota-se que para apenas 5000 pedidos, são necessários cerca de 54 segundos e 450 MB de memória RAM.

O número de restrições e vinculações é sempre 4 e o número de retrocessos é sempre 0. O custo, o tempo de execução, memória utilizada, *prunnings* e *resumptions* aumenta exponencialmente com os dados de entrada (de notar que o gráfico apresentado está em escala logarítmica).

10 Conclusões e Perspetivas de Desenvolvimento

A solução que aqui apresentamos é bastante limitada, sem qualquer correspondência a uma situação real. O enunciado do problema original não é realizável em Prolog com recurso aos módulos de PLR, tanto por questões técnicas como também é limitado à quantidade de informação que é capaz de processar, não conseguindo apresentar resultados em tempo de útil quando são usados dados de entrada semelhantes ao que era de esperar num sistema *cloud*.

Não foram feitas referências a outros autores nem trabalhos porque os problemas que são feitos nesta área não correspondem ao problema que aqui apresentamos.

A Anexos

A.1 Código Fonte

```
:- use_module(library(lists)).
:- use_module(library(random)).
:- use_module(library(clpfd)).
:- use_module(library(samsort)).

config_file('tp2_config.pl').

read_config :-
    config_file(F),
    open(F, read, Str),
    read_file(Str, _),
    close(Str).

read_file(Stream, []) :-
    at_end_of_stream(Stream).
read_file(Stream, [X|L]) :-
    \+ at_end_of_stream(Stream),
    read(Stream, X),
    assert(X),
    read_file(Stream, L).

all_equal([], _).
all_equal([X|T], X) :-
    all_equal(T, X).

generate_task_types(L, AuthR, ScriptsR, StorageR) :-
    requests(R),
    auth_percentage(AP),
    scripts_percentage(SP),
    storage_percentage(StP),
    AuthR is round(R * (AP / 100)),
    ScriptsR is round(R * (SP / 100)),
    StorageR is round(R * (StP / 100)),
    length(AuthL, AuthR),
    length(ScriptsL, ScriptsR),
    length(StorageL, StorageR),
    all_equal(AuthL, 0),
    all_equal(ScriptsL, 1),
    all_equal(StorageL, 2),
    append([AuthL, ScriptsL, StorageL], List),
    random_permutation(List, L).

highest_request_time(Time) :-
    auth_request_time(Auth), storage_request_time(Storage),
    script_request_time(Script),
    max_member(Time, [Auth, Storage, Script]).
```

```
46
47 %% indexed by machine id (0 - auth, 1 - storage, 2 - script)
48 request_time(0, X) :- auth_request_time(X).
49 request_time(1, X) :- storage_request_time(X).
50 request_time(2, X) :- script_request_time(X).
51
52 request_disk(0, X) :- auth_request_disk(X).
53 request_disk(1, X) :- storage_request_disk(X).
54 request_disk(2, X) :- script_request_disk(X).
55
56 request_ram(0, X) :- auth_request_ram(X).
57 request_ram(1, X) :- storage_request_ram(X).
58 request_ram(2, X) :- script_request_ram(X).
59
60 request_cpu(0, X) :- auth_request_cpu(X).
61 request_cpu(1, X) :- storage_request_cpu(X).
62 request_cpu(2, X) :- script_request_cpu(X).
63
64 generate_tasks([], [], [], [], [], []).
65 generate_tasks([TaskType|TaskTypes], [TaskDisk|TaskDisks], [
       TaskRAM|TaskRAMs], [TaskCPU|TaskCPUs], [TaskST|TaskSTs], [
       TaskET|TaskETs]) :-
66     request_time(TaskType, Time),
67     request_disk(TaskType, Disk),
68     request_ram(TaskType, RAM),
69     request_cpu(TaskType, CPU),
70     TaskDisk = task(TaskST, Time, TaskET, Disk, TaskType),
71     TaskRAM = task(TaskST, Time, TaskET, RAM, TaskType),
72     TaskCPU = task(TaskST, Time, TaskET, CPU, TaskType),
73     generate_tasks(TaskTypes, TaskDisks, TaskRAMs, TaskCPUs,
       TaskSTs, TaskETs).
74
75 generate_machines_disk([machine(0, A), machine(1, B), machine
       (2, C)]) :-
76     server_auth_disk(A),
77     server_storage_disk(B),
78     server_script_disk(C).
79
80 generate_machines_ram([machine(0, A), machine(1, B), machine
       (2, C)]) :-
81     server_auth_ram(A),
82     server_storage_ram(B),
83     server_script_ram(C).
84
85 generate_machines_cpu([machine(0, A), machine(1, B), machine
       (2, C)]) :-
86     server_auth_cpu(A),
87     server_storage_cpu(B),
88     server_script_cpu(C).
89
```

```
90 calculate_cost_queues(_, [], _, TempSlots, TempSlots).
91 calculate_cost_queues([StartTime|StartTimes], [NextTime|
       NextTimes], OldestTime, TempSlots, TotalSlots) :-
92     DT is NextTime - StartTime,
93     calc_time_and_slots(StartTime, OldestTime, DT,
       NewOldestTime, TempSlots, NewSlots),
94     calculate_cost_queues(StartTimes, NextTimes, NewOldestTime
       , NewSlots, TotalSlots).
95
96 calc_time_and_slots(StartTime, OldestTime, DT, StartTime,
       TempSlots, TempSlots) :-
97     routing_spd_time(RouteTime),
98     highest_request_time(ReqTime),
99     DT >= RouteTime,
100    DTOld is StartTime - OldestTime,
101    DTOld >= ReqTime.
102
103 calc_time_and_slots(StartTime, OldestTime, DT, OldestTime,
       TempSlots, TempSlots) :-
104    routing_spd_time(RouteTime),
105    highest_request_time(ReqTime),
106    DT >= RouteTime,
107    DTOld is StartTime - OldestTime,
108    DTOld < ReqTime.
109
110 calc_time_and_slots(StartTime, OldestTime, DT, OldestTime,
       TempSlots, NewSlots) :-
111    routing_spd_time(RouteTime),
112    highest_request_time(ReqTime),
113    DT < RouteTime,
114    DTOld is StartTime - OldestTime,
115    DTOld < ReqTime,
116    NewSlots is TempSlots + 1.
117
118 calc_time_and_slots(StartTime, OldestTime, DT, StartTime,
       TempSlots, NewSlots) :-
119    routing_spd_time(RouteTime),
120    highest_request_time(ReqTime),
121    DT < RouteTime,
122    DTOld is StartTime - OldestTime,
123    DTOld >= ReqTime,
124    NewSlots is TempSlots + 1.
125
126 ordered_start_times([], TempList, List) :-
127    samsort(TempList, List). %% samsort does not remove
       duplicates
128 ordered_start_times([Task|Tasks], TempList, List) :-
129    arg(1, Task, ST), %% task(ST, DT, ET, Resc, Mach), arg is
       1-indexed
130    append(TempList, [ST], TempList1),
```

```
131     ordered_start_times(Tasks, TempList1, List).
132
133 euro_unicode(8364).
134
135 write_profit_or_loss(X) :-
136     X > 0,
137     write('+'), write(X).
138 write_profit_or_loss(X) :-
139     write(X).
140
141 init :-
142     read_config,
143     generate_task_types(Types, AuthR, StorageR, ScriptsR),
144     requests(R),
145     length(TaskDisks, R), length(TaskRAMs, R), length(TaskCPUs
      , R), %% needed?
146     length(TaskSTs, R), length(TaskETs, R),
147     generate_tasks(Types, TaskDisks, TaskRAMs, TaskCPUs,
      TaskSTs, TaskETs),
148     generate_machines_disk(MachineDisks),
149     generate_machines_ram(MachineRAMs),
150     generate_machines_cpu(MachineCPUs),
151     routing_spd_time(RouteTime),
152     MaxTaskTime is 1000*60*60*24,
153     domain(TaskSTs, RouteTime, MaxTaskTime),
154     domain(TaskETs, RouteTime, MaxTaskTime),
155     domain([End], RouteTime, MaxTaskTime),
156     maximum(End, TaskETs),
157     cumulatives(TaskDisks, MachineDisks, [bound(upper)]),
158     cumulatives(TaskRAMs, MachineRAMs, [bound(upper)]),
159     cumulatives(TaskCPUs, MachineCPUs, [bound(upper)]),
160     append(TaskSTs, [End], Vars),
161     labeling([], Vars), %% program doesn't run if we use [
      minimize(End)]
162     write('====== FD Statistics ======'), nl,
163     fd_statistics, nl,
164     ordered_start_times(TaskDisks, [], [StartTime|StartTimes])
      ,
165     calculate_cost_queues([StartTime|StartTimes], StartTimes,
      StartTime, 0, Slots),
166     write('========= Statistics ======='), nl,
167     statistics, nl,
168     spd_slot_price(SlotPrice),
169     server_auth_price(AuthPrice),
170     server_storage_price(StoragePrice),
171     server_script_price(ScriptPrice),
172     Cost is AuthPrice + StoragePrice + ScriptPrice + (Slots *
      SlotPrice), %% SPD slots only
173     budget(Budget),
174     euro_unicode(Euro),
```

```
175     ProfitLoss is Budget - Cost,
176     write('========== Results =========='), nl,
177     write('Number of Authentication Requests: '), write(AuthR)
        , nl,
178     write('Number of Storage Requests: '), write(StorageR), nl
        ,
179     write('Number of Script Execution Requests: '), write(
        ScriptsR), nl,
180     write('Number of SPD slots bought: '), write(Slots), nl,
181     write('Number of SSD slots bought: '), write(0), nl,
182     write('Number of Authentication Servers: '), write(1), nl,
183     write('Number of Storage Servers: '), write(1), nl,
184     write('Number of Script Execution Servers: '), write(1),
        nl,
185     write('Available budget: '), write(Budget), put_code(Euro)
        , nl,
186     write('Total cost: '), write(Cost), put_code(Euro), nl,
187     write('Profit/Loss: '), write_profit_or_loss(ProfitLoss),
        put_code(Euro), nl.
```

tp2.pl

A.2 Ficheiro de Configuração

```
1  % base config
2  server_auth_disk(120).
3  server_auth_ram(100).
4  server_auth_cpu(200).
5  server_storage_disk(4000).
6  server_storage_ram(85).
7  server_storage_cpu(75).
8  server_script_disk(70).
9  server_script_ram(400).
10 server_script_cpu(1000).
11
12 % hardware
13 spd_slot_price(50).
14 ssd_slot_price(30).
15 server_auth_price(1000).
16 server_storage_price(1800).
17 server_script_price(2800).
18 disk_unit(250).
19 ram_unit(400).
20 cpu_unit(300).
21 disk_price(100).
22 ram_price(250).
23 cpu_price(420).
24
25 % tasks
26 routing_spd_time(100).
```

```
routing_ssd_time(250).
delegation_time(50).
auth_request_time(400).
storage_request_time(850).
script_request_time(100).

% resources
auth_request_disk(50).
auth_request_ram(80).
auth_request_cpu(75).
storage_request_disk(120).
storage_request_ram(40).
storage_request_cpu(30).
script_request_disk(30).
script_request_ram(150).
script_request_cpu(220).

% other
scripts_to_auth_servers_ratio(2).
budget(20000).
requests(5000).
auth_percentage(25).
scripts_percentage(60).
storage_percentage(15).
```

tp2_config.pl