Um Algoritmo Paralelo para Priorização de Testes Baseada em Similaridade usando OpenMPI

Carlos Diego Nascimento Damasceno, Paulo S. L. Souza, Adenilso Simao
Instituto de Ciências Matemáticas e de Computação – ICMC
Universidade de São Paulo – USP
Av. Trabalhador São-carlense, 400 – 13566-590 – São Carlos – SP – Brasil
damascenodiego@usp.br, {pssouza,adenilso}@icmc.usp.br

ERAD-SP

Resumo

- Priorização de testes baseada em similaridade usa funções de similaridade para eficientemente ordenar testes, maximizando sua diversidade [1, 3].
- Testes parecidos tendem a detectar defeitos semelhantes de um sistema, logo sua execução simultânea não traz ganhos [1].
- Calculo de *matriz de similaridade* (*SM*) contendo os graus de similaridade entre todos os pares de teste (custo de $O(n^2)$).
- Ordenação de testes: algoritmo de priorização usando distância máxima local (LMDP [2]).

Algoritmo paralelo de geração de matrizes de similaridade (PGSM)

- Propomos um algoritmo paralelo de geração de matrizes de similaridade (**PGSM**) que usa *np* processos OpenMPI.
- Um processo *mestre* distribui os *nr* casos de teste entre os $np-1$ processos *escravos*.
- Escravos calculam uma parcela contígua de *SM* com no máximo $\frac{nr(nr-1)}{2(np-1)}$ similaridades.

Ex.: Alocação da tarefa de cálculo similaridade (d_s) entre três escravos ($p_{slv}, 1 \leq slv \leq 3$) e quatro casos de teste (i.e. $np = 4$).

$$\begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{bmatrix} 0 & d_s(t_1,t_2) \in p_1 & d_s(t_1,t_3) \in p_1 & d_s(t_1,t_4) \in p_2 \\ 0 & 0 & d_s(t_2,t_3) \in p_2 & d_s(t_2,t_4) \in p_3 \\ 0 & 0 & 0 & d_s(t_3,t_4) \in p_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Algoritmo *Parallel LMDP* (PLMDP)

- *AllReduce*: Busca o processo p com o par de testes $\langle t_i, t_j \rangle$ mais distinto
- *Broadcast*: Transmite o par de testes $\langle t_i, t_j \rangle$ para solicitar descarte

```
1: INPUT: T = {t1, t2, ..., tnr}, np − 1 escravos com seus SMp
2: OUTPUT: TCS     // Testes priorizados
3: TCS ← []
4: while #T > 0 do
5:     if #T > 1 then
6:         AllReduce(ds, p, MAXLOC) // max(ds(ti, tj)) ∈ SMp
7:         Broadcast(ti, tj, p) // p broadcast ⟨ti, tj⟩
8:         TCS.add(ti); TCS.add(tj)
9:         T ← T\{ti, tj} // mestre e escravos
10:    else
11:        TCS.add(ti) where ti ∈ T
12:        T ← ∅
13:    end if
14: end while
```

Resultados

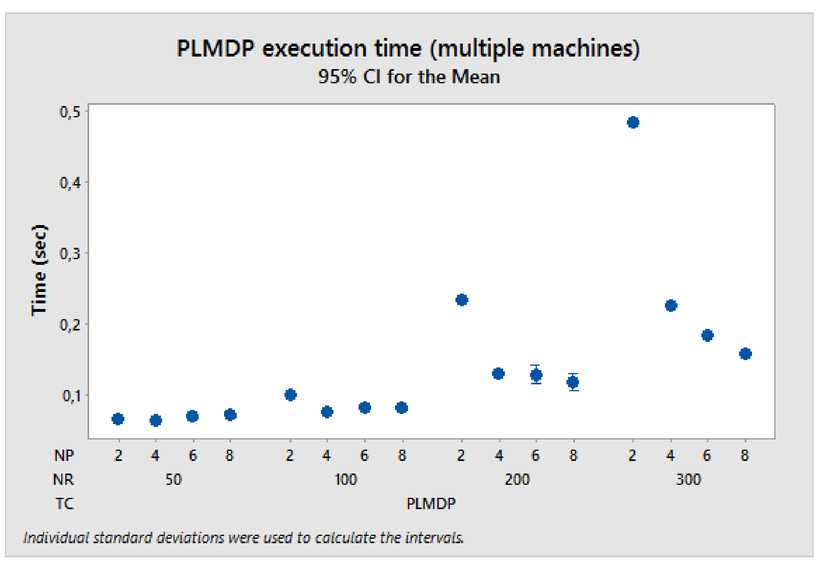

Figura 1: Tempo médio de execução do PLMDP

- *NP* $\Longrightarrow$ número de cálculos de similaridade
- *NP* e $avg(\Delta t)$ inversamente proporcionais
- Redução no Δt

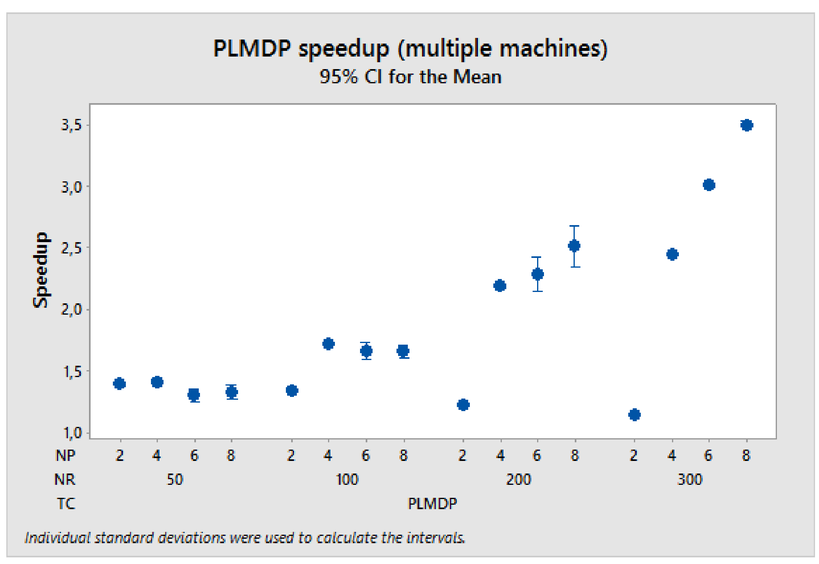

Figura 2: Speedup do PLMDP

- Speedup crescente em função de *NP*
- Speedup crescente para $NR > 2$
- Sobrecarga para $NR \leq 2$

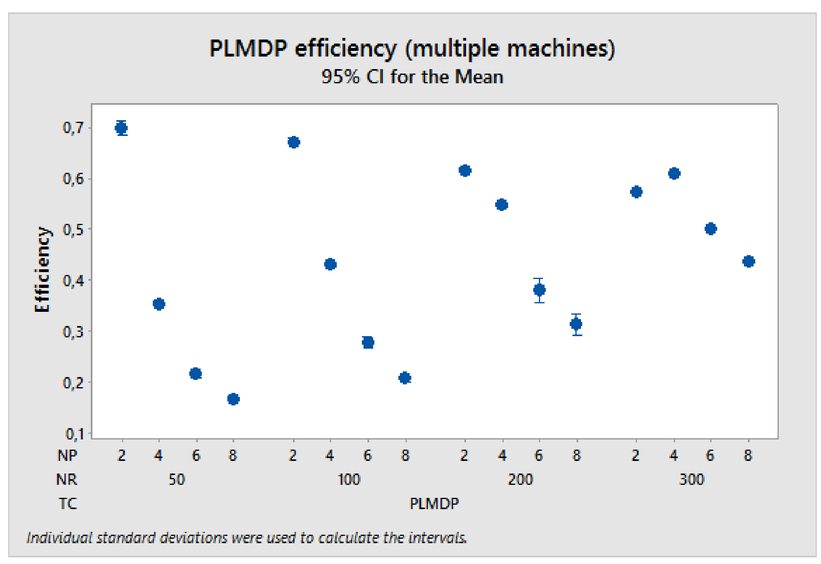

Figura 3: Eficiência do PLMDP

- PGSM $\Longrightarrow$ Baixa eficiência
- Escravos ociosos (*overhead*)
- Remoção de *pares mais distintos*

Experimento

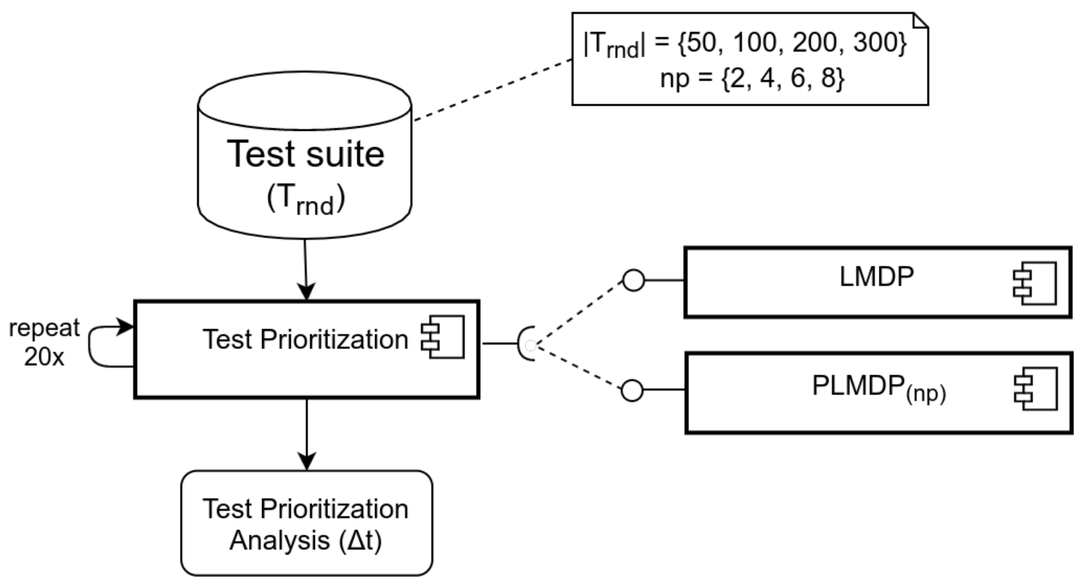

Figura 4: Avaliação do algorítmo PLMDP

Plataforma: Cluster com 8 nós Intel Core i7-4790, 32Gb RAM, 500Gb HD
Sistema operacional: Ubuntu 14.04 64 bits
Tecnologias: C++ (GCC 4.8.4) e OpenMPI (1.8.3)
Métricas: speedup = $\frac{\Delta t_{serial}}{\Delta t_{parallel}}$ e efficiency = $\frac{\Delta t_{serial}}{\Delta t_{parallel} \times np}$.
URL: `https://github.com/damascenodiego/fsmPrioritization`

Considerações finais

- Priorização paralela de testes baseada em similaridade
  PGSM: Distribuição dos pares de teste
  PLMDP: Busca paralela e ordenação dos pares mais distintos
- Implementação em C++ e OpenMPI
- *Mestre+Escravo:* um mestre e $np-1$ escravos
  Mestre: Distribui os testes entre os $np-1$ escravos
  Escravos: Calcula graus de similaridade em paralelo
- Resultados
  - Redução no tempo médio de execução
  - Queda no $avg(\Delta t)$ para *NP* maiores
  - Ineficiência no gerenciamento de recursos

Referências

[1] Emanuela G. Cartaxo, Patrícia D. L. Machado, and Francisco G. Oliveira Neto. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verif. and Reliab.*, 2011.

[2] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, Patrick Heymans, and Yves Le Traon. Bypassing the combinatorial explosion: Using similarity to generate and prioritize t-wise test configurations for software product lines. *IEEE Transactions on Software Engineering*, 40(7):650–670, 2014.

[3] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: A survey. *Softw. Test. Verif. Reliab.*, 22(2):67–120, March 2012.