Implementação de Testes de Escalonabilidade para Rate Monotonic

Allan Carlos Figueiredo Echeverria*, Vicente Knihs Erbs*
*Universidade Federal de Santa Catarina (UFSC)

I. Introdução

O Rate Monotonic (RM) é um algoritmo de escalonamento de prioridade fixa para sistemas operacionais de tempo real (RTOS), cujo critério para definir as prioridades consiste em favorecer aquelas tarefas que possuem menor período. Também, é importante notar que este é recomendado para situações nas quais o deadline e o período são os mesmos.

Quando um conjunto de tarefas é gerado, pode-se aplicar um ou mais métodos para verificar se tal conjunto pode ser escalonado pelo RM, chamados testes de escalonabilidade. Estes podem ser exatos (são precisos quanto a escalonabilidade ou não do conjunto), suficientes (precisos quando o teste afirma ser escalonável, mas imprecisos caso contrário) ou necessários (precisos ao afirmar que não é escalonável).

O presente docmento traz a implementação de dois testes suficientes - Liu & Layland e Hyperbolic Bound Function - e um exato - Response Time Analysis. Em seguida, os testes são avaliados sob diferentes situações e comparados entre si.

II. Implementação

Para a realização dos testes, a linguagem de programação Python foi escolhida. O principal fator para a escolha é devido às bibliotecas disponíveis, que facilitam a geração de dados, necessárias para criar as amostras a serem utilizadas nos testes, como também auxiliam na produção dos gráficos, necessários para a avaliação dos resultados obtidos no processo.

A estrutura geral do teste consiste na produção de 9 casos diferentes, produzidos pela combinação dos possíveis níveis dos parâmetros para a geração de tarefas, apresentados na Tabela I.

Caso de Uso	Utilização		Período [ms]	
	Mínimo	Máximo	Mínimo	Máximo
Light	0.0001	0.01	3	33
Moderate	0.001	0.09	10	100
Heavy	0.09	0.1	50	250

Tabela I: Limites das variáveis

Então, cada combinação de paramêtros passa por 10 testes, referentes ao uso máximo do sistema. A extensão desses testes vai de 10% à 100% de uso, ao passo de variação de 10% por iteração. Em cada iteração são gerados 100 conjuntos de tarefas, cada qual com um número aleatório de tarefas, tendo como restrição para a produção de tarefas a combinação de parâmetros (que limitam a utilização e o período de cada tarefa) e o somatório das tarefas do conjunto deve ser igual à utilização máxima do sistema.

Tomando como exemplo a combinação 1 (*light utilization* e *light period*) e o uso máximo de 10%, serão gerados 100 conjuntos de tarefas para esse teste. Cada um desses 100 conjuntos tem como certeza que cada uma das tarefas terá a utilização compreendida no limite **[0.001;0.01]** e período compreendido no limite **[3ms;33ms]**. Além disso, a soma da utilização de todas as tarefas será igual a 0.1.

A. Main

As variáveis utilizadas no programa estão dispostas na Tabela II.

Variável	Uso
f	Arquivo utilizado para armazenar os conjuntos de testes utilizados;
graphH, graphL e graphR	Arrays que armazenam os valores que serão plotados nos gráficos de cada teste;
countH, countL e countR	Variáveis utilizadas para armazenar a taxa de sucesso para cada algoritmo de teste.

Tabela II: Váriáveis utilizadas na implementação

Para realizar essa tarefa, foi utilizado um conjunto de *nested loops* em 3 níveis. Os níveis correspondem, do mais abrangente ao mais exclusivo: combinação de paramêtros, máxima utilização e conjunto de tarefas.

Ao fim de cada subloop de geração de conjunto de tarefas, o conjunto é ordenado de forma crescente, registrado no arquivo de armazenamento e passa pelos 3 testes em sequência. Então, a variável *count* respectiva a cada teste é incrementada em caso de sucesso.

Em sequência, com a finalização de cada subloop de variação da máxima utilização do sistema, a taxa de sucesso é armazenada no array do respectivo algoritmo.

Então, ao fim de cada loop das combinações de parâmetros, o gráfico referente à específica combinação é gerado e plotado na pasta raíz.

B. generateTaskSet

Esta função é responsável por gerar um conjunto de tarefas, a partir dos parâmetros recebidos: *max_utilization* (referente à utilização máxima do sistema) e *case* (referente à combinação específica de parâmetros de tarefas).

A função trata a geração de tarefas de duas formas diferentes: em casos de utilização do parâmetro **Heavy Utilization**, há algumas limitações quanto ao uso de diferentes tarefas. Para este caso, é verificado se a utilização máxima é inferior à 0.9, e, havendo validação positiva, a única solução possível são com todas as tarefas com utilização igual a 0.1. Se a validação for negativa, existem 2 possibilidades: a de gerar todas as tarefas iguais, como no exemplo anterior, ou todas as tarefas com utilização mínima e uma tarefa com utilização máxima.

No segundo caso, onde a combinação não utiliza o caso **Heavy Utilization**, a variedade de tarefas é maior. Sendo assim, o único tratamento especial para a produção de tarefas ocorre quando o valor mínimo do caso de utilização somado à utilização atual ultrapassa a utilização máxima do sistema. Nesse caso, o valor atribuído à última tarefa é a diferença entre a utilização atual e a utilização máxima.

Por fim, não há limitações na produção dos períodos das tarefas. Nesse sentido, as tarefas são geradas aleatoriamente a partir dos limites do parâmetro de período do caso.

C. Hyperbolic Bound Function

O teste foi implementado na função *hyperbolic_test*, que recebe como parâmetro um array com as utilizações de um conjunto de tarefas. Segundo a definição fornecida por Buttazzo[1], um conjunto de *i* tarefas, cada qual com uma utilização U_i, é escalonável caso a seguinte condição for verdadeira:

$$\prod_{i=1}^{n}(U_i + 1) \leq 2 \tag{1}$$

Para verificação de tal o produtório é representado como um laço for percorrendo todos os itens do array. Em seguida a função retorna 1 em caso de sucesso e 0 em caso de falha no escalonamento.

D. Sufficient Schedulability Test

Proposto por Chang L. Liu e James W. Layland em 1973[2], neste programa foi implementado na função *liu_lay_test*. Segundo o método, um determinado conjunto de tarefas é escalonável caso a seguinte condição seja satisfeita:

$$\sum_{i=1}^{n} U \leq n(2^{\frac{1}{n}} - 1) \tag{2}$$

onde *U* é a utilização do sistema e *n* é o número de tarefas no conjunto.

Para este caso, o somatório foi representado como um loop *for*, onde, além de realizar o somatório, verifica se o conjunto é harmônico, ou seja, todos os períodos são divisíveis pelo menor período do conjunto. Essa verificação é feita pois, caso o conjunto for harmônico, o somatório das utilizações deve ser menor ou igual a um. Caso o período não for harmônico, ele passa pelo teste equacionado acima, e, em caso de sucesso é retornado 1, caso contrário, o retorno é 0.

E. Response Time Analysis

Esse algoritmo, implementado na função *rta_test*, foi proposto originalmente por Audsley[3]. O algoritmo consiste na implementação de duas equações, sendo a Equação 3 utilizada apenas para a primeira iteração:

$$R_i^{(0)} = C_i + \sum_{j=1}^{i-1} C_j \tag{3}$$

E, para as demais iterações, realizadas a um passo *k*, o algoritmo segue a Equação 4

$$R_i^{(k)} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{`}(k-1)}{T_j} \right\rceil C_j \tag{4}$$

No contexto dessas equações, R_i representa o pior caso de resposta de uma tarefa *i* dentro de um determinado conjunto. C_i é o tempo de computação dessa tarefa, expresso pelo produto entre o período e a utilização do sistema pela tarefa. O somatório de ambas as equações tem como função percorrer todas as tarefas de maior prioridade do conjunto, fato que é garantido pela ordenação decrescente conforme a prioridade das tarefas.

O macro loop que percorre as tarefas é realizado na ordem inversa da lista, isto é, da menor para a maior prioridade. Isso se deve ao fato de tarefas de menor prioridade estarem mais sucetíveis a falharem no teste, uma vez que todas as tarefas de maior prioridade podem retardar seu tempo de resposta.

A escalonabilidade de uma tarefa para teste consiste na satisfação da condição que o pior caso de resposta da tarefa testada seja menor que o período disponível para execução da tarefa. Assim sendo, a única condição que retorna falha na escalonagem do conjunto de tarefas é quando o resultado final do tempo de resposta de uma tarefa do conjunto é superior ao período de execução disponível.

III. Resultados

Os resultados obtidos para cada combinação estão expostos nas figuras de 1 a 9.

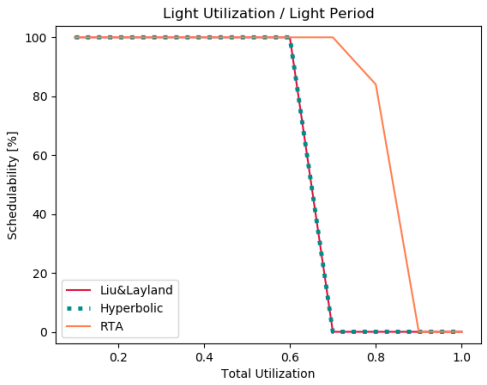

Figura 1: Light Utilization e Light Period

A partir dos gráficos, é possível perceber que os testes suficientes acabam por não escalonar muitos conjuntos quando

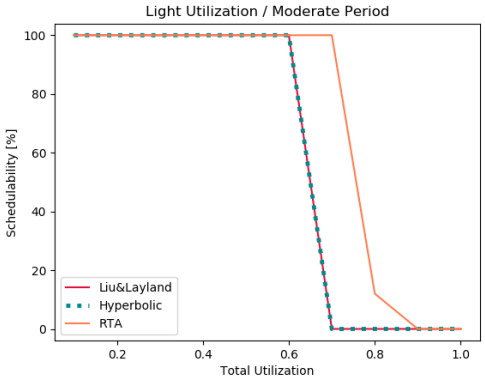

Figura 2: Light Utilization e Moderate Period

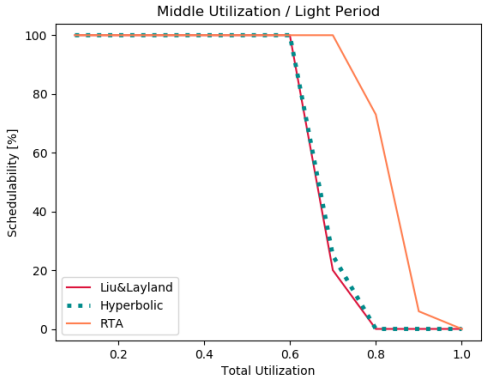

Figura 4: Middle Utilization e Light Period

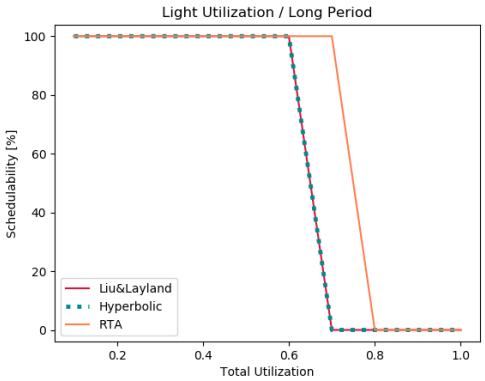

Figura 3: Light Utilization e Long Period

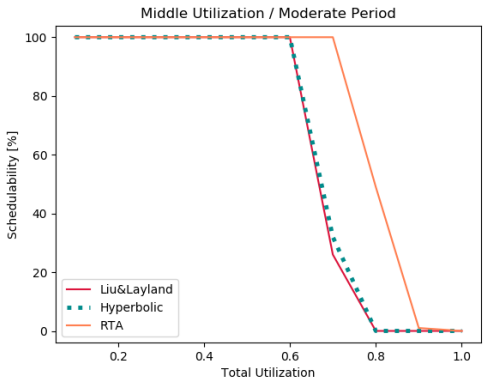

Figura 5: Middle Utilization e Moderate Period

a utilização do sistema é superior ou igual a 70% - o que era esperado matematicamente, dada a formulação dos testes. Apesar disso, ambos apresentam bons resultados quando a utilização total não ultrapassa esse limite.

Na maioria dos casos, os testes suficientes trazem exatamente os mesmos valores, mas quando as tarefas têm utilização média, o Hyperbolic Bound Function traz resultados menos pessimistas e, portanto, mais precisos que o Liu & Layland. Já que o teste além de mais preciso é também mais simples, o Hyperbolic é recomendado sobre o Liu & Layland.

Em termos de complexidade, o RTA é certamente o que mais exige cálculos para obter conclusões. Todavia, este traz a maior precisão, já que é um teste exato. Sob essa perspectiva, esse teste é o mais apropriado para casos em que a utilização total do conjunto das tarefas seja maior ou igual a 0.7, uma vez que essa região não é favorável ao uso dos testes suficientes.

IV. Conclusão ou Conclusões

Desse modo, foi possível trazer a implementação dos testes de escalonabilidade Liu & Layland, Hyperbolic Bound Function e RTA e os avaliar com conjuntos de tarefas de características distintas. Percebeu-se que é possível cobrir todas as faixas de utilização total do sistema apenas com dois testes: o Hyperbolic para casos abaixo de 0.7 e o RTA para as outras condições. Além disso, não houveram grandes diferenças nos resultados para as diferentes combinações, de modo que o teste mais apropriado para cada situação não depende desse fator.

Referências

[1] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Springer Publishing Company, Incorporated, 2011.

[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, p. 46–61, Jan. 1973. [Online]. Available: https://doi-org.ez46.periodicos.capes.gov.br/10.1145/321738.321743

[3] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

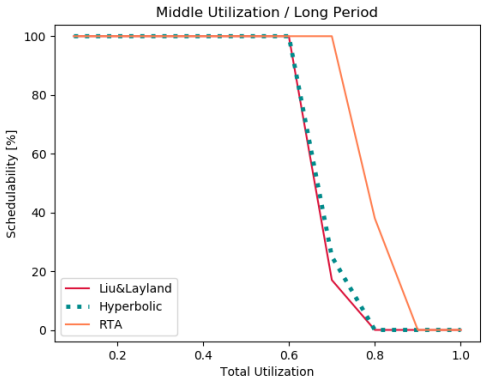

Figura 6: Middle Utilization e Long Period

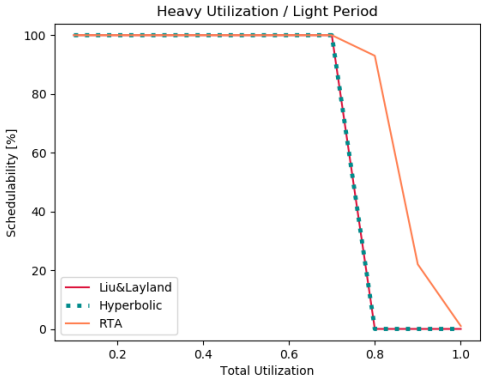

Figura 7: Heavy Utilization e Light Period

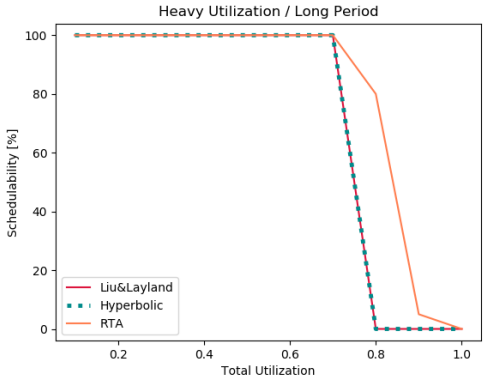

Figura 9: Heavy Utilization e Long Period

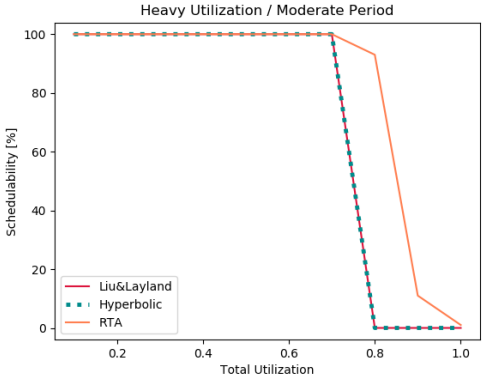

Figura 8: Heavy Utilization e Moderate Period