Método Empírico para Avaliar a Sensibilidade do Tempo de Execução de Tarefas de Tempo Real aos Dados de Entrada

Karila Palma Silva, Luís Fernando Arcaro, Rômulo Silva de Oliveira
Departamento de Automação e Sistemas (DAS)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis-SC, Brasil
E-mail: { karila.palma, luis.arcaro }@posgrad.ufsc.br, romulo.deoliveira@ufsc.br

***Resumo*—Neste trabalho realizamos uma análise empírica dos tempos de execução de tarefas de tempo real com respeito aos dados de entrada. A análise tem por objetivo (1) verificar a sensibilidade dos tempos de execução de tarefas aos dados de entrada utilizados, e (2) avaliar quantitativamente seu impacto nos tempos de execução resultantes. Para (1) utilizamos testes estatísticos, onde verificamos se diferentes dados de entrada geram distribuições diferentes, podendo então concluir se existem evidências estatísticas de que o tempo de execução da tarefa é sensível aos dados de entrada. Com relação a (2) utilizamos um algoritmo genético para encontrar dados de entrada que maximizam (MAX) ou minimizam (MIN) o tempo de execução da tarefa, e calculamos a razão da mediana das entradas MAX e MIN. A finalidade da análise realizada é fornecer um método para o testador de *software* obter informações sobre o impacto dos dados de entrada nos tempos de execução da tarefa, e portanto, a importância da identificação dos dados de entrada de pior caso — com respeito ao tempo de execução — para serem utilizados no teste de tarefas em Sistemas de Tempo Real.**

I. Introdução

Os sistemas computacionais de tempo real são identificados como sistemas submetidos, além de requisitos de natureza lógica, a requisitos de natureza temporal, ou Sistemas de Tempo Real (STRs), onde os resultados devem estar corretos não somente do ponto de vista lógico, mas também devem ser gerados no momento correto. Os requisitos temporais a que esses sistemas estão sujeitos são expressos em termos dos prazos (*deadlines*) nos quais os resultados devem ser gerados. Testes de escalonabilidade são empregados para demonstrar que, mesmo no pior caso, as tarefas de um determinado sistema podem ser escalonadas cumprindo seus prazos. Esses testes requerem, além do conhecimento do período e do *deadline* que são impostos ao sistema pelo contexto de operação, a determinação do tempo máximo de execução no pior caso — *Worst-Case Execution Time* (*WCET*) — de cada uma das tarefas do sistema, ou de um limite para ele, sendo esses derivados considerando o *software* e o *hardware* utilizados [1], [2], [3].

Para a determinação de limites para *WCETs* existem tanto métodos estáticos quanto métodos baseados em medição. Os métodos estáticos geram limites seguros por meio de uma análise detalhada do código da tarefa e da arquitetura do *hardware*, o que implica em grandes esforços de modelagem e computacionais. Os métodos baseados em medição analisam os tempos de execução das tarefas efetivamente produzidos durante a execução. Tais métodos reduzem significativamente os esforços de análise, mas exigem a determinação de margens de segurança para considerar possíveis efeitos de temporização não observados, e exigem amostras de tempos de execução representativas em relação ao pior cenário dos dados de entrada e do contexto de execução utilizado [4], [2], [5]. Este trabalho está relacionado ao uso de métodos baseados em medição, e tem por objetivo fornecer métodos para o testador de *software* obter informações sobre o impacto — e, portanto, o esforço que deve ser colocado na identificação — dos dados de entrada de pior caso da tarefa com respeito ao tempo de execução.

Uma tarefa de *software* pode ter tempos de execução diferentes quando executada múltiplas vezes em uma mesma plataforma de *hardware*. As principais fontes de variabilidade temporal são (1) o *hardware* do processador usado, e (2) os caminhos de execução que são efetivamente medidos. Com respeito a (1) os tempos de execução são afetados pelas latências dos elementos internos do processador que realizam a execução das instruções. A introdução de elementos de aceleração — por exemplo, *pipeline*, *caches* de instruções e dados e *branch prediction* — para melhorar o desempenho, diminui os tempos de execução, mas os torna variáveis e dependentes do histórico de execução (ou seja, o histórico de execução precisa ser levado em conta para prever o estado do processador no início da execução de uma instrução específica). Em relação a (2) encontrar o(s) pior(es) caminho(s) de tempo de execução — *Worst-Case Execution Paths* (*WCEPs*) — e garantir que estes tenham sido testados é um grande desafio. O que define qual caminho é executado são as variáveis de entrada e as variáveis permanentes alteradas em execuções anteriores, sendo que alguns estados de variáveis permanentes são atingidos somente após muitas execuções da tarefa. Portanto, várias execuções de uma tarefa podem produzir tempos diferentes devido às características do *hardware* e às entradas utilizadas nos testes [6], [7], [8], [9].

Testes exaustivos de todos os caminhos de execução com todas as possíveis combinações de dados de entrada são geralmente impraticáveis. A forma mais fácil de gerar dados de

entrada é a aleatória, porém essa não tem um bom desempenho em termos de cobertura. Portanto, é necessária a utilização de métodos analíticos ou heurísticos para derivar os *WCEPs*, considerando os limites de iteração dos laços e possíveis caminhos divergentes (*if-then-else*) e o fluxo do programa [2], [10], [11].

Nesse contexto, propomos uma análise de sensibilidade e de impacto dos dados de entrada com respeito ao tempo de execução de tarefas que compõem STRs. A classificação da tarefa em relação à sensibilidade aos dados de entrada pode fornecer informações importantes para determinar a quantidade de caminhos a serem executados. Se há evidências de que a tarefa não é sensível aos dados de entrada, ou seja, gera a mesma distribuição de tempo quando executada com diferentes dados de entrada, a exploração de muitos caminhos não irá trazer benefícios à análise. Por outro lado, se a tarefa é sensível aos dados de entrada torna-se necessária a busca pelo caminho que leva a tempos de execução extremos.

Portanto, neste trabalho (1) realizamos um estudo empírico da sensibilidade dos tempos de execução de tarefas aos dados de entrada, (2) elaboramos um algoritmo genético destinado a encontrar dados de entrada que maximizam ou minimizam o tempo de execução de tarefas sensíveis aos dados de entrada, e (3) avaliamos seu impacto no tempo de execução considerando os dados de entrada obtidos em (2). Com isso, a finalidade deste trabalho é fornecer métodos para o testador de *software* obter informações sobre o impacto e a importância dos dados de entrada utilizados no teste de tarefas em STRs.

Este artigo está estruturado da seguinte maneira. A Seção II apresenta uma breve revisão de trabalhos relacionados. O ambiente e as condições experimentais são descritos na Seção III. A Seção IV apresenta os objetivos dos experimentos realizados. Os resultados são apresentados na Seção V, e finalmente a Seção VI resume as conclusões.

II. Trabalhos relacionados

A determinação dos dados de entrada que levam tarefas a apresentarem o tempo de execução de pior caso (*WCET*) ainda é considerada um problema em aberto [11]. Em geral, os dados de entrada de pior caso não são conhecidos e são de difícil derivação [2].

Técnicas como execução simbólica [12], [13] e análise de valor [2] são utilizadas para derivar dados de entrada, porém utilizam a modelagem do *software* e da arquitetura do *hardware* para análise. A alta complexidade dos processadores atuais afeta a aplicabilidade dessas técnicas. O comportamento temporal de um *software* que executa em processadores complexos, como o apresentado neste trabalho, é difícil de modelar, devido ao fato que o tempo de execução de uma instrução pode depender do histórico de execução. Portanto, através dessas técnicas não é possível determinar entradas que maximizam ou minimizam o tempo de execução da tarefa e, consequentemente, obter informações sobre o impacto e a importância dos dados de entrada utilizados no teste de tarefas em STRs.

Métodos heurísticos como algoritmos evolutivos são sugeridos como uma maneira rápida de encontrar casos de teste [14], [15], [16], [17]. Em [18] temos uma investigação sobre como algoritmos genéticos podem ser usados para estimar os tempos de execução mínimos e máximos de *softwares* utilizados em sistemas embarcados, onde o número de ciclos de *clock* do processador foi usado como função de aptidão/adequação. Em experimentos usando algoritmos genéticos, esses trabalhos foram capazes de identificar caminhos mais longos e mais curtos do que aqueles encontrados usando tanto testes aleatórios quanto testes sistemáticos.

Nosso trabalho difere principalmente dos trabalhos apresentados nesta seção por (1) analisar a sensibilidade aos dados de entrada através de testes estatísticos, e (2) analisar o impacto dos dados de entrada no tempo de execução da tarefa, com base nos dados de entrada que maximizam e minimizam o tempo de execução obtidos com um algoritmo genético. Até onde sabemos, não existe nenhum trabalho anterior que realize tais análises sobre dados de entrada de tarefas de tempo real.

III. Ambiente de experimentação

Nas próximas seções apresenta-se detalhes sobre o ambiente utilizado nos experimentos realizados, e a forma como procedeu-se para a coleta dos dados.

A. Plataforma

Os testes foram realizados em um microcomputador equipado com um processador Intel (R) Core (TM) i7-4770 executando o sistema operacional Ubuntu 16.04 (na versão 4.13.0 do *kernel* vanilla). O Ubuntu é um sistema operacional baseado na distribuição Debian Linux, distribuída como *software* livre e de código aberto. O sistema foi iniciado no nível de execução cinco (*runlevel* N 5), ou seja, modo multiusuário que permite que o sistema execute todos os serviços e interface gráfica, o nível de execução padrão para a maioria dos sistemas *desktop* baseados em Linux. Destacamos que o processador usa *hyper-threading*, o que significa que o agendador de instruções, que emite instruções fora de ordem, mistura instruções de várias *threads* para criar um *mix* de instruções que reduz o tempo médio de execução. Isso torna o tempo de execução de uma parte específica do código não-determinista, porque depende das outras tarefas executadas ao mesmo tempo. Em sistemas de tempo real, considera-se boa prática a desabilitação de *hyper-threading* e a fixação da frequência do processador, para reduzir seus efeitos nos tempos de execução. Portanto, em nossos testes desabilitamos *hyper-threading* e fixamos a política de escala de frequência do processador, para que as avaliações propostas fossem realizadas em um cenário mais apropriado para aplicações de tempo real [19].

B. Tarefa

Escolhemos quatro tarefas dos pacotes de *Benchmarks Mälardalen* [20] e *TACLeBench* [21] para serem analisadas neste trabalho, sendo elas:

- *bsort*, que ordena um vetor de elementos usando o método *bubble sort*.
- *cnt*, que conta números não-negativos em uma matriz.

- *dijkstra*, que encontra o caminho mais curto entre nós em um grafo.
- *qurt*, que realiza a computação da raiz de equações quadráticas.

Cada tarefa foi implementada como função em uma *thread* que repete a execução a cada 10 milissegundos para *bsort*, *cnt* e *qurt*, e 50 milissegundos para *dijkstra*. Em outras palavras, definimos o número de execuções da tarefa (tamanho da amostra), geramos os dados de entrada, e coletamos uma medida a cada 10 ou 50 milissegundos dependendo da tarefa. Destacamos que (1) o processo é carregado apenas uma vez na memória e (2) todos os tempos de execução são medidos em microssegundos (μs).

C. Coleta de Dados

Os tempos de execução analisados foram medidos a partir de contadores de *hardware* usando a ferramenta Perf. Perf é uma ferramenta de *profiling* para sistemas baseados em Linux 2.6+ que abstrai as diferenças de *hardware* da Unidade Central de Processamento (UCP) nas medições de desempenho do Linux. Ele é baseado na interface *perf_events*, exportada por versões recentes do *kernel* do Linux. A ferramenta Perf é uma coleção integrada de subcomandos que permite inspeção e análise em vários níveis [22].

Neste trabalho, executamos o comando *perf record* — como exemplificado abaixo — definindo um conjunto de pontos de rastreio dinâmicos da tarefa através do comando *perf probe* [19]. Também utilizamos eventos de rastreamento para as interrupções externas, *Interrupt Request (IRQ)* e *Non-Maskable Interrupt (NMI)*, para filtrar o tempo de execução da tarefa sob análise. O *IRQ* é um pedido de interrupção enviado do nível de *hardware* para a UCP. Ao receber a interrupção, a UCP alterna para um contexto de interrupção para tratar o evento sinalizado. A *NMI* é um tipo de interrupção que difere do mecanismo padrão por não ser mascarável, forçando portanto seu atendimento por parte do processador [23].

```
sudo perf record
-e probe_bsort:bsort_task
-e probe_bsort:bsort_task_return
-e irq:irq_handler_entry
-e irq:irq_handler_exit
-e nmi:nmi_handler
-e irq:softirq_entry
-e irq:softirq_exit
-e sched:sched_switch
-- filter ``prev_comm == bsort ||
next_comm == bsort'' -a
```

As amostras coletadas através do comando *perf record* são salvas em um arquivo binário chamado, por padrão, *perf.data*. Usamos o comando *perf script* para ler o *perf.data* e produzir o *trace* do qual extraímos o tempo de execução da tarefa.

Usamos a ferramenta Perf para deduzir a interferência direta de outras tarefas e tratadores de interrupção. Essa dedução pode não ser completa, devido à complexidade do sistema-alvo e ao comando Perf utilizado. Em nossos testes, encontramos evidências de que os chaveamentos de contexto no Linux têm um grande efeito em processadores complexos, por exemplo, causando interferência indireta via *pipeline*, *cache*, *Translation Lookaside Buffer (TLB)*, etc. Essa interferência indireta não foi removida dos tempos de execução medidos. No entanto, esse é um método simples e que pode ser amplamente aplicado, e mostrou-se suficiente para os objetivos deste trabalho.

IV. Objetivos dos experimentos

Neste trabalho, realizamos uma análise empírica a respeito dos dados de entrada das tarefas. Os experimentos que realizamos consistem em (1) verificar se a tarefa é sensível aos dados de entrada com respeito ao tempo de execução, (2) obter dados de entrada que maximizem ou minimizem o tempo de execução, e (3) avaliar o impacto dos dados de entrada sobre o tempo de execução. Para (1) serão utilizados os testes estatísticos apresentados na Seção IV-A. Em relação a (2) utilizaremos o algoritmo genético apresentado na Seção IV-B. Para (3) coletaremos grandes amostras para os dados de entrada obtidos em (2), e realizaremos a análise apresentada na Seção IV-C.

A. Testes Estatísticos

Utilizamos testes estatísticos de hipóteses para mostrar evidência de distribuição idêntica (d.i.). Os testes estatísticos de hipóteses são baseados em uma hipótese (H_0) a ser testada que é considerada verdadeira a menos que evidências sejam encontradas para refutá-la, e uma hipótese alternativa (H_1) que é aceita se e somente se H_0 é rejeitada através de evidências adequadas. Testes de hipóteses geralmente produzem p-valores que estão associados à probabilidade da estatística de teste fornecer um resultado tão longe do esperado quanto o observado na amostra. Eles são então usados para avaliar a possibilidade de um determinado resultado ser produzido unicamente por acaso. Esses testes estão sujeitos a erros dos tipos "falso negativo" e "falso positivo" em relação a H_0, que devem, portanto, ser controlados para a tomada de decisões com base nos resultados produzidos. Quando apenas pequenas amostras podem ser obtidas, isso é feito através da determinação de um limite α para a probabilidade de falsos negativos, conhecido como nível de significância, cujo valor típico está entre 0.05 e 0.01. Ao comparar o p-valor p de um teste com α, toma-se a decisão de rejeitar H_0 se $p < \alpha$ ou não rejeitá-lo se $p \geq \alpha$, com um nível de confiança dado por $\gamma = 1 - \alpha$.

Neste trabalho, para controlar os erros dos testes de hipótese, empregamos amostras grandes e as dividimos em um conjunto de segmentos escolhidos aleatoriamente — usamos 100 segmentos, cada um com tamanho 100. Nossas conclusões são então derivadas com base no comportamento estatístico dos resultados obtidos, levando em conta que os testes que empregamos produzem p-valores uniformemente distribuídos ou com tendência a valores altos no intervalo $[0, 1)$ quando H_0 de fato é verdadeira [24]. Os resultados são apresentados na forma de um gráfico *box and whisker* que destaca os quantis 0%, 5%, 50%, 95% e 100% dos p-valores obtidos, ou seja, o mínimo, a mediana, o máximo e os quantis 5% e 95%. Essa

abordagem proporciona maior confiança nos resultados dos testes estatísticos empregados, uma vez que a probabilidade de testemunhar falsos resultados tende a diminuir à medida que tais testes são replicados.

Para evidenciar distribuição idêntica (d.i.), usamos os testes estatísticos Kolmogorov-Smirnov (KS) e k-sample Anderson-Darling (AD). O teste KS tem como hipótese nula que as observações seguem a mesma distribuição, e produz um p-valor que está relacionado à distância entre as distribuições empíricas das amostras avaliadas [25], [26]. O teste k-sample AD está disponível em duas versões, as quais chamamos de AD1 e AD2, que diferem principalmente pela função de distribuição empírica usada, uma ajustada para tamanhos de amostra possivelmente diferentes, e outra que enfatiza diferenças na cauda das distribuições [27].

Através dos p-valores produzidos pelos testes estatísticos empregados, podemos concluir que (1) não existem fortes evidências de que os valores observados apresentam diferentes distribuições, quando a aplicação desses testes produz p-valores que sejam uniformemente distribuídos ou apresentem tendência a valores altos no intervalo $[0, 1)$, ou que (2) existem evidências de que os valores observados aderem a distribuições diferentes entre si, quando a aplicação desses testes produz p-valores que apresentem tendência a valores próximos a zero.

Destacamos que os testes estatísticos apresentados foram aplicados utilizando o *software* estatístico R [28].

B. Algoritmo Genético (AG)

Para as tarefas que são sensíveis aos dados de entrada utilizamos um algoritmo genético para encontrar as entradas que minimizam ou maximizam o tempo de execução da tarefa. Destacamos que os dados de entrada obtidos através do algoritmo genético avaliado não são necessariamente os que levam ao (1) tempo mínimo de execução no melhor caso — também conhecido como *Best-Case Execution Time* (*BCET*), ou ao (2) tempo máximo de execução no pior caso — *WCET*, devido à complexidade do ambiente de experimentação utilizado — isto é, computador e o sistema operacional.

O algoritmo genético considerado foi implementado conforme proposto em [18]. Realizamos testes com diferentes configurações de indivíduos e gerações que são possíveis em tempo factível. Para a finalidade deste trabalho, consideramos satisfatória a seguinte configuração:

- População inicial: Inicializamos a população aleatoriamente, sendo essa composta por dez indivíduos — dez dados de entrada gerados aleatoriamente.
- Função de aptidão/adequação: Avaliamos a aptidão dos indivíduos com base no tempo de execução. Destacamos que, o ambiente de experimentação utilizado gera grande variabilidade temporal. Portanto, para minimizar este efeito, consideramos como aptidão do indivíduo a mediana de três execuções da tarefa.
- Gerações: Definimos 10000 gerações, sendo que em cada geração cinco novos indivíduos são gerados e cada um substituirá algum indivíduo da população se sua aptidão (1) for maior quando queremos o dado de entrada que maximiza o tempo de execução, ou (2) menor quando queremos o dado de entrada que minimiza o tempo de execução.
- Ao final de todas as gerações, coletamos uma amostra de tamanho 100 para cada um dos dez dados de entradas resultantes para verificar qual realmente gera tempos de execuções maiores para (1) e menores para (2).

C. Análise do Impacto dos Dados de Entrada (IDE)

A análise do impacto dos dados de entrada (IDE) empregada neste trabalho tem por objetivo avaliar o efeito que a variação de um dado de entrada pode ocasionar no tempo de execução resultante da tarefa. Quando existem evidências de que uma pequena variação no dado de entrada altera significativamente o tempo de execução, diz-se que a tarefa é sensível aos dados de entrada. Neste contexto, buscamos encontrar percentuais do impacto do dado de entrada no tempo de execução de tarefas.

Para calcular o IDE, (1) fixamos os dados de entrada que maximizam (MAX) e minimizam (MIN) o tempos de execução da tarefa, (2) coletamos uma amostra representativa para as entradas MAX e MIN, e (3) calculamos a razão da mediana (MED) de ambas as amostras conforme a Equação 1. Consideramos a mediana, e não o maior e o menor valor observado, com o objetivo de minimizar o efeito do ambiente de experimentação no tempo de execução da tarefa.

$$IDE = \frac{MED\_MAX}{MED\_MIN} \tag{1}$$

O percentual do IDE é dado pela Equação 2:

$$(IDE - 1) * 100 \tag{2}$$

Através do fornecimento do IDE, podemos prover informações ao testador de *software* sobre o impacto que os dados de entrada têm sobre o tempo de execução da tarefa.

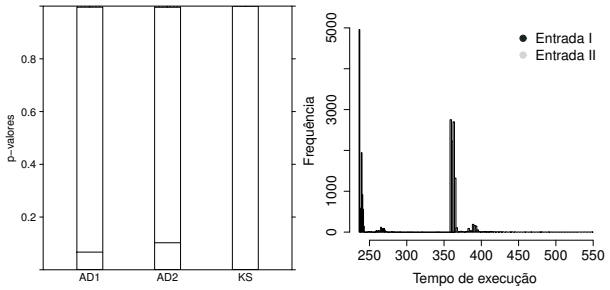
A fim de avaliar a variabilidade à qual o IDE está sujeito, efetuamos seu cálculo sobre amostras cujo tamanho é aumentado de 10^3 até 10^5 com incrementos de tamanho 10^3. Ou seja, calculamos o IDE considerando as 10^3 primeiras medições, posteriormente, ele é obtido considerando essas 10^3 medições mais um incremento de 10^3, totalizando $2 \cdot 10^3$ medições, e assim até que 10^5 medições sejam utilizadas, produzindo-se assim 100 valores do IDE ($100 * 10^3 = 10^5$). A observação de pequena variação nos valores obtidos para o IDE considerando amostras de tamanho crescente permite concluir que amostras relativamente pequenas de cada uma das entradas MAX e MIN são suficientes para aplicação do método pelo testador de *software* em situações práticas.

V. Resultados experimentais

A. Análise de Sensibilidade aos Dados de Entrada (SDE)

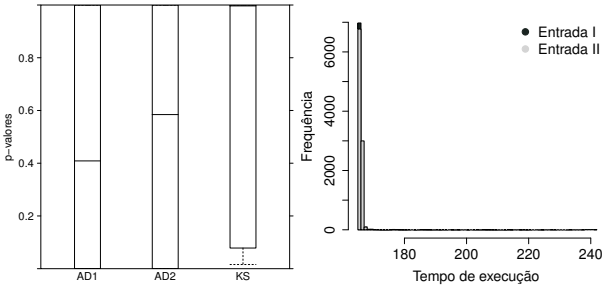
Para verificar se a tarefa é sensível aos dados de entrada, usamos os testes estatísticos KS e k-sample AD apresentados na Seção IV-A, os quais são empregados para evidenciar distribuição idêntica (d.i.). As Figuras 1, 2, 3 e 4 apresentam o histograma dos tempos de execução das tarefas avaliadas para

duas diferentes entradas, e o gráfico dos p-valores produzidos pelos testes estatísticos de distribuição idêntica. A partir de sua análise, podemos concluir que (1) as tarefas *cnt* e *qurt* executadas com diferentes valores de entrada geram distribuições idênticas, uma vez que os p-valores produzidos são aproximadamente uniformemente distribuídos ou apresentam tendência a valores altos, e que (2) as tarefas *bsort* e *dijkstra* executadas com diferentes dados de entrada geram distribuições diferentes, uma vez que os p-valores produzidos apresentam tendência a valores baixos.



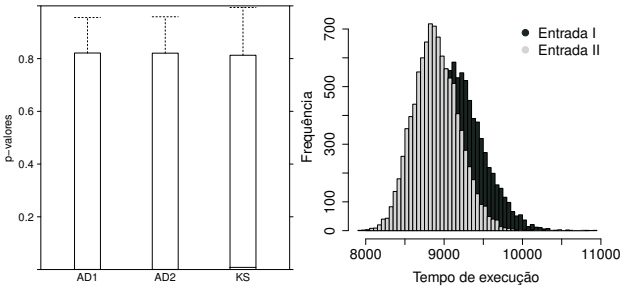
(a) Teste d.i. (b) Histograma

Figura 1. SDE - *bsort*



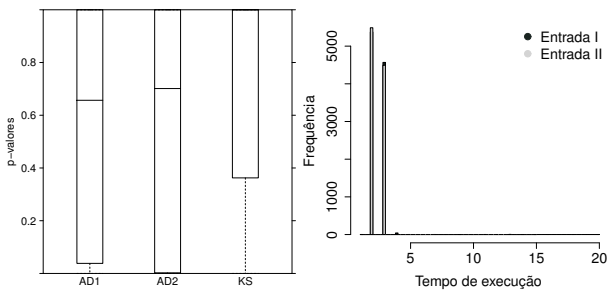
(a) Teste d.i. (b) Histograma

Figura 2. SDE - *cnt*



(a) Teste d.i. (b) Histograma

Figura 3. SDE - *dijkstra*



(a) Teste d.i. (b) Histograma

Figura 4. SDE - *qurt*

Para tarefas que não são sensíveis aos dados de entrada, (1) a exploração de muitos dados de entrada não trará resultados diferentes, e (2) toda a variabilidade do tempo de execução decorre do ambiente e não do dado de entrada, portanto, não faz sentido calcular o IDE. Por outro lado, se a tarefa é sensível aos dados de entrada, torna-se necessário encontrar dados de entrada que minimizam ou maximizam o tempo de execução da tarefa para podermos calcular o IDE.

B. Dados de Entrada para Teste

Nesta seção apresentamos como foram obtidos os dados de entrada utilizados na análise do IDE (Seção V-C).

bsort: Sensível aos dados de entrada. Ressaltamos que essa tarefa ordena um vetor de 500 elementos usando o método *bubble sort*. Essa tarefa é frequentemente usada na análise temporal para obter o (1) *WCET* com dados de entrada fixados como um vetor inteiro de ordem inversa (chamamos de Invertido), porque é o caminho de execução que executa o maior número possível de operações elementares, e (2) *BCET* com dados de entrada fixados como um vetor inteiro ordenado (chamamos de Ordenado), porque é o caminho de execução que executa o menor número possível de operações elementares.

Nas Tabelas I e II apresentamos o tempo de execução das entradas, obtidos na geração final do algoritmo genético — conforme apresentado na Seção IV-B — que minimizam e maximizam o tempo de execução da tarefa, respectivamente.

Tabela I
TEMPOS DE EXECUÇÃO MIN OBTIDO PELO AG

Entrada	Tempo de execução (μs)
I_{Min}	380
II_{Min}	380
III_{Min}	380
IV_{Min}	379
V_{Min}	380
VI_{Min}	380
VII_{Min}	379
$VIII_{Min}$	378
IX_{Min}	379
X_{Min}	379

Tabela II
TEMPOS DE EXECUÇÃO MAX OBTIDOS PELO AG

Entrada	Tempo de execução (μs)
I_{Max}	550
II_{Max}	536
III_{Max}	537
IV_{Max}	535
V_{Max}	551
VI_{Max}	548
VII_{Max}	576
$VIII_{Max}$	543
IX_{Max}	592
X_{Max}	548

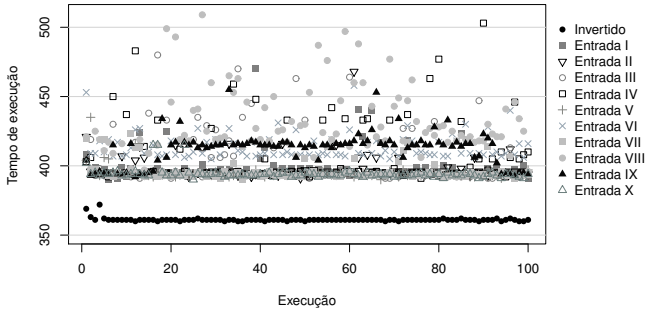


Figura 6. *bsort* - Max

Na Figura 5 apresentamos 100 execuções de cada uma das dez entradas que minimizam o tempo de execução da tarefa obtidas através do algoritmo genético e a entrada Ordenado. Observa-se que a entrada Ordenado gera tempos de execução menores. Portanto, como entrada que minimiza o tempo de execução vamos considerar a entrada Ordenado para a análise que segue. Similarmente, na Figura 6 apresentamos para as entradas que maximizam o tempo de execução da tarefa obtidas através do algoritmo genético e a entrada Invertido. Observa-se que a entrada Invertido gera tempos de execução menores. Como o objetivo é considerar a entrada que maximiza o tempo de execução, vamos considerar a entrada VIII para a análise que segue. A entrada Invertido não é o pior caso pelo fato de que arquiteturas de computador complexas empregam mecanismos especulativos, como memórias *cache* e *branch prediction*, que podem gerar latências menores quando um mesmo conjunto de laços é executado. Por essa razão, os dados de entrada que levam a um grande número de operações elementares, como o Invertido, podem produzir tempos de execução menores do que dados de entrada que requerem a execução de menos operações elementares.

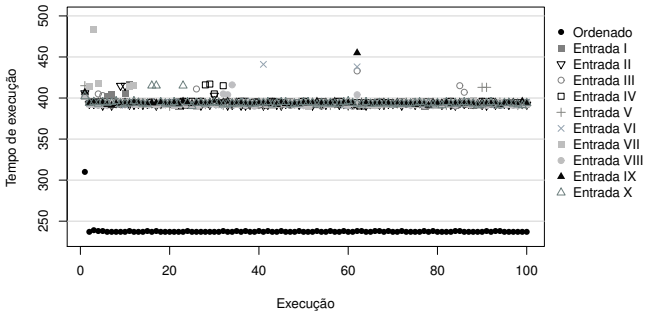


Figura 5. *bsort* - Min

dijkstra: Também é sensível aos dados de entrada. Essa tarefa consiste em encontrar o caminho mais curto entre nós em um grafo. Em [21] é sugerida uma entrada para ser utilizada na análise do *WCET* (chamamos de Original). Acreditamos que a entrada que leva ao *BCET* seja o grafo composto por um único valor, porque é o caminho que executa o menor número possível de operações elementares (chamamos de Fixa). No entanto, como não temos certeza de quais dados de entrada minimizam e maximizam o tempo de execução da tarefa, aplicamos o algoritmo genético — conforme apresentado na Seção IV-B.

Nas Tabelas III e IV apresentamos o tempo de execução das entradas que, respectivamente, minimizam e maximizam o tempo de execução da tarefa obtidos na geração final do algoritmo genético.

Tabela III
TEMPOS DE EXECUÇÃO MIN OBTIDO PELO AG

Entrada	Tempo de execução (μs)
I_{Min}	8075
II_{Min}	8124
III_{Min}	8046
IV_{Min}	8122
V_{Min}	8035
VI_{Min}	7928
VII_{Min}	8082
$VIII_{Min}$	8052
IX_{Min}	8090
X_{Min}	8034

Tabela IV
TEMPOS DE EXECUÇÃO MAX OBTIDOS PELO AG

Entrada	Tempo de execução (μs)
I_{Max}	10467
II_{Max}	10371
III_{Max}	10456
IV_{Max}	10380
V_{Max}	10451
VI_{Max}	10543
VII_{Max}	10395
$VIII_{Max}$	10473
IX_{Max}	10405
X_{Max}	10529

Na Figura 7 apresentamos 100 execuções de cada uma das dez entradas que minimizam o tempo de execução da tarefa obtidas através do algoritmo genético e a entrada Fixa. Observa-se que a entrada Fixa gera tempos de execução menores.

Portanto, como entrada que minimiza o tempo de execução vamos considerar a entrada Fixa para a análise que segue. Na Figura 8 apresentamos informações similares para as entradas que maximizam o tempo de execução da tarefa obtidas através do algoritmo genético e a entrada Original. Observa-se que a entrada Original gera tempos de execução menores. Como o objetivo é considerar a entrada que maximiza o tempo de execução, vamos considerar a entrada VII na análise que segue.

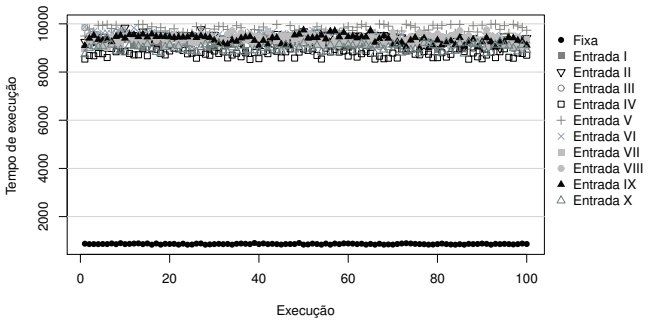


Figura 7. *dijkstra* - Min

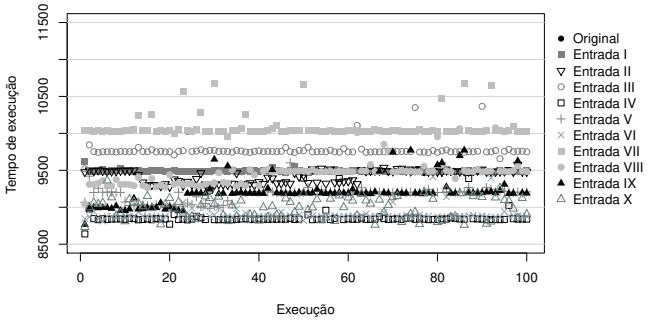


Figura 8. *dijkstra* - Max

C. Impacto dos Dados de Entrada (IDE)

Dado que foram definidas as entradas MIN e MAX para cada tarefa sensível aos dados de entrada, elas serão usadas para computar o IDE de cada tarefa, conforme o método descrito na Seção IV-C. Os resultados obtidos para as tarefas *bsort* e *dijkstra* são apresentados, respectivamente, nas Figuras 9 e 10. Podemos observar que para a tarefa *bsort* o IDE foi $\approx 1,645$ ($\approx 64\%$) e não houve alteração no seu valor para os diferentes tamanhos da amostra analisados. Para a tarefa *dijkstra* o IDE foi $\approx 11,25$ ($\approx 1025\%$) e houve uma variação muito pequena (< 0.1) no seu valor para os diferentes tamanhos da amostra analisados. Portanto, conclui-se que (1) o testador de *software* deve definir cuidadosamente os dados de entrada para serem utilizados na análise temporal das tarefas *bsort* e *dijkstra*, o que não é necessário com as tarefas *cnt* e *qurt*, (2) o valor do IDE é consistente, uma vez que não foram observadas variações significativas no seu valor conforme o tamanho da amostra é aumentado de 10^3 até 10^5 medições, portanto, uma amostra de tamanho 10^3 de cada uma das entradas MAX e MIN é suficiente para uso em situações práticas, e (3) o IDE da tarefa *dijkstra* é muito superior ao da tarefa *bsort*, devido tanto à grande influência que os dados de entrada utilizados para teste tem sobre seu tempo de execução quanto à sua maior complexidade computacional.

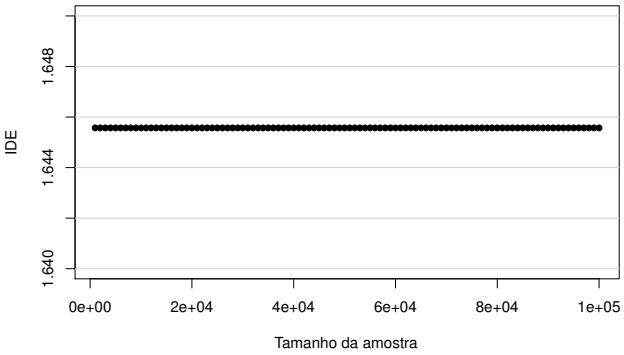


Figura 9. IDE - *bsort*

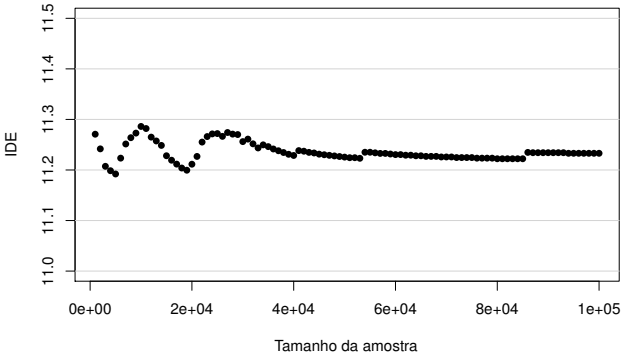


Figura 10. IDE - *dijkstra*

VI. Conclusão

Neste trabalho apresentamos um estudo empírico da sensibilidade dos tempos de execução de tarefas de tempo real aos dados de entrada utilizados, e avaliamos seu impacto nos tempos de execução resultantes. Na análise de sensibilidade medimos o tempo de execução da tarefa com diferentes dados de entrada, e empregamos testes estatísticos para verificar se esses exercem influência sobre o tempo de execução da tarefa, ou seja, se levam a distribuições de tempos de execução diferentes. Para a análise do impacto do dado de entrada calculamos a razão da mediana para os dados de entrada que maximizam e que minimizam o tempo de execução da tarefa. Para encontrar essas entradas utilizamos um algoritmo genético. Destacamos que, dada a natureza do computador e sistema operacional que forma a plataforma computacional empregada neste trabalho, existe grande variância no tempo de execução e, portanto, os verdadeiros *WCETs* e *BCETs* das tarefas avaliadas permanecem desconhecidos. No entanto, o algoritmo genético foi capaz de encontrar dados de entrada que geram tempos de execução maiores que aqueles utilizados na análise do *WCET* das tarefas avaliadas em situações práticas.

A partir das análises apresentadas, podemos concluir que as tarefas *bsort* e *dijkstra* apresentam evidências de alta

sensibilidade aos dados de entrada. Já para as tarefas *cnt* e *qurt*, não existem evidências de que são sensíveis aos dados de entrada. O método apresentado tem por objetivo fornecer informações para o testador de *software* sobre o impacto e, portanto, o esforço que deve ser colocado na identificação dos dados de entrada de pior caso da tarefa com respeito ao tempo de execução. Tarefas que apresentam evidências de sensibilidade aos dados de entrada exigem um esforço maior em testes para a identificação dos dados de entrada de pior caso com respeito ao tempo de execução, a fim de, garantir que o(s) *WCEP*(*s*) tenham sido executados. Para tarefas que não apresentam evidências de sensibilidade aos dados de entrada, não há necessidade de que muitos caminhos sejam executados. Portanto, ressaltamos a importância da análise de sensibilidade e de impacto dos dados de entrada, que compõem o método proposto neste trabalho, para orientar os esforços dos testes com respeito ao tempo de execução de tarefas em STRs.

AGRADECIMENTO

Esta pesquisa foi financiada pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e pela CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior).

REFERÊNCIAS

[1] J. W.-S. Liu, *Real-Time systems*, 1st ed. Prentice Hall, 2000.

[2] R. Wilhelm, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, and R. Heckmann, "The Worst-Case Execution-Time Problem - Overview of Methods and Survey of Tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, pp. 36:1–36:53, 2008.

[3] G. C. Buttazzo, "Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications," *Vasa*, 2008.

[4] F. J. Cazorla, J. Abella, J. Andersson *et al.*, "PROXIMA: Improving Measurement-Based Timing Analysis through Randomisation and Probabilistic Analysis," in *Euromicro Conference on Digital System Design 2016 (DSD'16)*. IEEE, 2016, pp. 276–285.

[5] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-askasua, J. Perez, E. Mezzetti, and T. Vardanega, "WCET Analysis Methods: Pitfalls and Challenges on their Trustworthiness," in *International Symposium on Industrial Embedded Systems 2015 (SIES'15)*. IEEE, 2015, pp. 1–10.

[6] J. Engblom, "Processor pipelines and static worst-case execution time analysis," Ph.D. dissertation, Uppsala University, 2002.

[7] J. Schneider, "Cache and pipeline sensitive fixed priority scheduling for preemptive real-time systems," in *Real-Time Systems Symposium 2000 (RTSS'00)*. IEEE, 2000, pp. 195–204.

[8] N. Zhang, A. Burns, and M. Nicholson, "Pipelined processors and worst case execution times," *Real-Time Systems*, vol. 5, pp. 319–343, 1993.

[9] J. Yan and W. Zhang, "WCET Analysis for Multi-Core Processors with Shared L2 Instruction Caches," in *Real-Time and Embedded Technology and Applications Symposium 2008 (RTAS'08)*. IEEE, 2008, pp. 80–89.

[10] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand, "Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, pp. 966–978, 2009.

[11] S. Law and I. Bate, "Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis," in *Euromicro Conference on Real-Time Systems 2016 (ECRTS'16)*. IEEE, 2016, pp. 189–199.

[12] J. Edvardsson, "A survey on automatic test data generation," in *Proceedings of the 2nd Conference on Computer Science and Engineering*, 1999, pp. 21–28.

[13] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later," *Communications of the ACM*, vol. 56, no. 2, pp. 82–90, 2013.

[14] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based worst-case execution time analysis," in *Software Technologies for Future Embedded and Ubiquitous Systems, 2005. SEUS 2005. Third IEEE Workshop on*. IEEE, 2005, pp. 7–10.

[15] L. C. Briand, Y. Labiche, and M. Shousha, "Stress testing real-time systems with genetic algorithms," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*. ACM, 2005, pp. 1021–1028.

[16] J. Wegener and F. Mueller, "A comparison of static analysis and evolutionary testing for the verification of timing constraints," *Real-Time Systems*, vol. 21, no. 3, pp. 241–268, 2001.

[17] I. Ashraf, G. M. Hassan, K. Yahya, S. A. Shah, S. Ullah, A. Manzoor, and M. Murad, "Parameter tuning of evolutionary algorithm by meta-eas for wcet analysis," in *Emerging Technologies (ICET), 2010 6th International Conference on*. IEEE, 2010, pp. 7–10.

[18] J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres, "Testing real-time systems using genetic algorithms," *Software Quality Journal*, vol. 6, no. 2, pp. 127–135, 1997.

[19] D. B. Oliveira and R. S. Oliveira, "Comparative Analysis of Trace Tools for Real-Time Linux," *IEEE Latin America Transactions*, vol. 12, no. 6, pp. 1134–1140, Sept 2014.

[20] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET Benchmarks: Past, Present And Future," in *International Workshop on Worst-Case Execution Time Analysis 2010 (WCET'10)*, vol. 15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.

[21] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, ser. OpenAccess Series in Informatics (OASIcs), M. Schoeberl, Ed., vol. 55. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016, pp. 2:1–2:10.

[22] Perf, "Linux Perf tool," 2015. [Online]. Available: https://perf.wiki.kernel.org/

[23] D. B. Oliveira and R. S. Oliveira, "Automata-based modeling of interrupts in the Linux PREEMPT RT kernel," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2017, pp. 1–8.

[24] G. Marsaglia and W. W. Tsang, "Some Difficult-to-pass Tests of Randomness," *Journal of Statistical Software*, vol. 7, pp. 1–9, 2002.

[25] J. Abella, E. Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla, "Heart of Gold: Making the Improbable Happen to Increase Confidence in MBPTA," in *Euromicro Conference on Real-Time Systems 2014 (ECRTS'14)*. IEEE, 2014, pp. 255–265.

[26] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, C. Hernandez, A. Gianarro, I. Broster, and F. J. Cazorla, "Fitting Processor Architectures for Measurement-Based Probabilistic Timing Analysis," *Microprocessors and Microsystems (MICPRO)*, vol. 47B, pp. 287–302, 2016.

[27] F. W. Scholz and M. A. Stephens, "K-Sample Anderson-Darling Tests," *Journal of the American Statistical Association*, vol. 82, 1987.

[28] R, "R: A Language and Environment for Statistical Computing," 2017. [Online]. Available: http://www.r-project.org/