Uma Abordagem para Estimar Tempos de Execução em Sistemas de Tempo Real baseados em Componentes*

Ricardo Perrone[1], Raimundo Macêdo[1], George Lima[1] , Verônica Lima[2]

[1]Laboratório de Sistemas Distribuídos - LaSiD
[1]Programa de Pós-Graduação em Mecatrônica

[1]Dep. de Ciência da Computação
[2]Dep. de Estatística
Universidade Federal da Bahia (UFBA)
Campus de Ondina, CEP:40170-110, Salvador-BA, Brazil

{perrones,macedo,gmlima,cadena}@ufba.br

Abstract. *Many component-based real-time systems have recently been proposed as a solution to modular and easily maintainable distributed real-time systems. This paper proposes a methodology for estimating probability distributions of execution times in the context of such systems, where no access to component internal code is assumed. In order to evaluate the proposed methodology, experiments were conducted with components implemented over CIAO with the goal to estimate a probability distribution. The collected experimental data show that the proposed approach is indeed a good approximation for component execution time probability distribution.*

Resumo. *Muitos sistemas baseados em componentes têm sido recentemente propostos como solução para sistemas distribuídos de tempo real de fácil manutenção e modulares. Este artigo propõe uma metodologia para a estimativa de uma distribuição de probabilidade de tempo de execução, no contexto de tais sistemas baseados em componentes. Nenhum acesso ao código interno do componente é assumido. A fim de avaliar a metodologia proposta, experimentos foram conduzidos com componentes desenvolvidos e instalados sobre CIAO, com o propósito de estimar uma distribuição de probabilidade. Os dados experimentais coletados mostram que a abordagem proposta é realmente uma boa aproximação para distribuição de probabilidade de tempos de execução de componente.*

1. Introdução

Os requisitos de sistemas distribuidos modernos, tais como maior flexibilidade, interoperabilidade e redução de custos, tem motivado o uso intensivo de soluções baseadas em *software* e a exploração de componentes COTS (*Commercial Off-The-Shelf*), tanto de *hadware* quanto de *software*. Neste contexto, os componentes de

*O segundo autor recebeu apoio do CNPq e FAPESB (projetos 478739/2006-0 e APR0089/2006, respectivamente) e o terceiro autor recebeu apoio do CNPq (projeto 475851/2006-4).

software distribuídos aparecem como uma tecnologia promissora, uma vez que sua abordagem modular e desacoplada permite o desenvolvimento e composição de componentes que ajudam na produção de sistemas reutilizáveis e de fácil manutenção. Por isso, o uso de *middleware* baseado em componentes de tempo real para a construção de sistemas distribuídos de tempo real tem merecido muita atenção da comunidade nestes últimos anos [Wang et al. 2004a].

As aplicações de um modo geral podem se beneficiar da adoção de soluções modulares produzidas a partir da composição de componentes COTS, desde que sua programação de código tenha total aderência aos requisitos da *interface* especificada. Contudo, a falta de conhecimento da estrutura de código interna do componente (abordagem *caixa-preta*) torna difícil a verificação das garantias temporais (*timeliness*) da composição de componentes que formam o sistema. Neste caso, diferentes configurações de composição podem produzir variações nas propriedades do sistema como um todo, o que torna a tarefa de verificação ainda mais difícil, e pode representar uma barreira para adoção dessa abordagem no contexto de tempo real.

Um maneira convencional de verificação de garantias temporais de um sistema de tempo real crítico é calcular a viabilidade de se cumprir o limite de tempo (*deadline*) de cada uma das tarefas relacionadas, mesmo quando todas as tarefas são ativadas concorrentemente. Para viabilizar esta análise, deve-se calcular primeiro o pior caso de tempo de execução (*Worst Case Execution Time* - WCET) de cada tarefa isoladamente, e então combinar tais WCETs em fórmulas que capturam o pior caso de tempo de resposta de cada tarefa [Audsley et al. 1993] ou a demanda máxima de processador [Liu and Layland 1973]. Contudo, para calcular precisamente o WCET de cada tarefa deve-se considerar o tempo de execução de cada instrução relativo ao pior caso do caminho de execução. Além disso, as variações dos tempos de execução das instruções devido à arquitetura de *hardware*, tais como memória *cache* e *pipelines*, também devem ser consideradas.

Infelizmente, quando componentes COTS são usados, nem sempre é possível aplicar tal abordagem de verificação, visto que o código fonte do componente não está sempre disponível. Então, sistemas de tempo real baseados em componentes COTS têm aplicabilidade limitada em sistemas críticos que requerem elevado nível de segurança, de tal modo que a perda de *deadlines* pode causar grandes prejuízos. Por outro lado, existem outros cenários de tempo real tais como multimídia, telecomunicações e algumas aplicações industriais, em que a perda de *deadlines* pode causar apenas uma degradação da qualidade de serviço, mas que é considerada tolerável dado que a probabilidade de tais perdas acontecerem está abaixo de um limite previamente determinado.

Este artigo aborda esse desafio ao propor e validar uma metodologia para a estimativa de uma distribuição de probabilidade de tempo de execução, no contexto de sistemas de tempo real distribuídos baseados em componentes. Esta estimativa pode ajudar projetistas a verificar o comportamento temporal de sistemas baseado em componentes, aplicando modelos alternativos de análises de desempenho [Kim et al. 2005, Manolache et al. 2001]. A motivação principal da abordagem proposta é aplicá-la para estimar o tempo de resposta de serviços desenvolvidos no

contexto do ARCOS (ARquitetura para COntrole e Supervisão), um *framework* baseado em componentes que foi desenvolvido sobre o CIAO e dedicado à construção de sistemas distribuídos de controle e supervisão de unidades industriais [Andrade and Macêdo 2007].

Para avaliar a metodologia proposta, experimentos foram conduzidos com componentes que foram conectados e instalados sobre a plataforma do CIAO, e sua distribuição de probabilidade foi então estimada. A análise dos dados coletados demonstrou que a abordagem proposta é de fato uma boa aproximação para a estimativa da distribuição de probabilidade de tempo de execução de um componente de software.

O restante deste artigo está estruturado da seguinte forma: Na seção 2 são discutidos os trabalhos relacionados ao tema. Na seção 3 é descrito o modelo de sistema e a arquitetura de componentes de tempo real utilizada como suporte aos experimentos. Já na seção 4 uma nova abordagem para a estimativa de uma distribuição de probabilidaddes de tempo de execução é apresentada. Na seção 5 um estudo de caso é ilustrado, e a eficácia da abordagem apresentada é avaliada. Finalmente, na seção 6 são apresentadas considerações finais sobre o trabalho realizado e perspectivas de trabalhos futuros.

2. Trabalhos relacionados

Muitos trabalhos no campo da análise de tempo para sistemas de tempo real são dedicados ao esforço de se calcular, com maior precisão possível, o WCET . Juntos, esses trabalhos apresentam diversas abordagens para tratar os fatores de influência (entrada de dados, caminhos de execução, memória *cache* e *pipeline*) que determinam a ocorrência do pior caso, e visam sobretudo reduzir o pessímismo e aumentar a precisão do WCET calculado. Contudo, dada a extrema dificuldade em se determinar o valor exato, o máximo que se consegue é uma estimativa do WCET que seja suficientemente segura para muitas aplicações de tempo real.

As técnicas relacionadas a este tema de estudo podem ser agrupadas nas seguintes categorias: Estática, Probabilística e Híbridas. A lista de publicações em análise estática é consideravelmente extensa, e concentram o foco em analisar o caminho de execução do código da apliação para poder derivar o valor de WCET de uma dada plataforma de *hardware*. Existem alguns trabalhos recentes que aplicam essa abordagem em sistemas de tempo real baseados em componentes [Estévez-Ayres et al. 2005, Ballabriga et al. 2007, Fredriksson 2006, Moss and Muller 2004]. Geralmente, as abordagens estáticas tendem a ser complexas e produzem estimativas pessimistas e/ou necessitam de acesso ao código fonte da aplicação. Isso motivou o desenvolvimento de abordagens que utilizam técnicas probabilísticas e híbridas para a geração de estimativas.

Um dos primeiros resultados sobre estimativas probabilísticas foi publicado por [Edgar and Burns 2001]. Após a realização de medições, os valores de tempo de execução obtidos de amostras foram utilizados para a derivação do WCET usando estatística aplicada ao estudo de valores extremos. Os autores assumiram que as amostras seguiam uma distribuição de Gumbel e eram independentes e identicamente distribuídas. Embora esta seja uma abordagem caixa-preta, ou seja, não

requer acesso e conhecimento sobre o código analisado, a suposição da independência dos dados pode limitar a aplicabilidade da abordagem para sistemas práticos. O trabalho realizado por [Bernat et al. 2002] pode ser visto como uma abordagem híbrida, composta por características provenientes de propostas estáticas e probabilísticas. Eles determinaram os perfis de tempos de execução de blocos de código (e.g. através de medições). Tais perfis constituem na realidade uma tabela contendo tempos de execução observados e suas respectivas frequências. Com isso, é possível se obter um valor de WCET e sua probabilidade associada, através da convolução e combinação dos perfis. Certos aspectos de dependêndia e independência das variáveis de tempo são levados em consideração, embora uma instrumentação refinada do código da aplicação tenha sido necessária. Um esquema para armazenar perfis de tempo de execução de sistemas baseados em componentes é também apresentado por [Nolte et al. 2003]. Eles argumentam que o componente do sistema poderia dispor de meios para monitorar a si mesmo. Baseado neste esquema, uma técnica de estimativa híbrida do WCET é proposta [Möller et al. 2005].

Ao invés de estimar um valor de WCET, este artigo foca na derivação de uma distribuição de probabilidade de tempo de execução em sistemas baseado em componentes. Tal distribuição tem aplicabilidade em modernos mecanismos de escalonamento de tempo real [Kim et al. 2005, Manolache et al. 2001]. Assim como em algumas técnicas probabilísticas, o método de estimativa proposto aqui é baseado em medições. Entretanto, não é requerido qualquer conhecimento prévio sobre o código interno da aplicação para a realização destas medições. A instrumentação do código foi utilizada apenas para efeito de avaliação final dos resultados obtidos. Nós supomos, por exemplo, que podem haver componentes projetados e desenvolvidos por equipes externas de desenvolvedores contratados. Então, o conhecimento do código da aplicação pode não estar disponível, o que faz das abordagens convencionais estáticas e híbridas inadequadas para este contexto. Além disso, diferentemente de algumas abordagens probabilísticas [Edgar and Burns 2001], nós não assumimos qualquer distribuição já conhecida e podemos considerar a adoção de sistemas complexos, mesmo que a independência dos dados não possa ser assegurada. O método proposto é, portanto, um método adequado ao modelo orientado a componentes COTS - versão compilada. Neste artigo apresentamos uma versão estendida dos resultados recentemente discutidos com a comunidade de tempo real [Perrone et al. 2008]. Os aspectos técnicos aqui descritos visam demonstrar com maior clareza e riqueza de detalhes a eficácia e eficiência do método proposto.

3. Arquitetura de Componentes

3.1. Modelo de Sistema

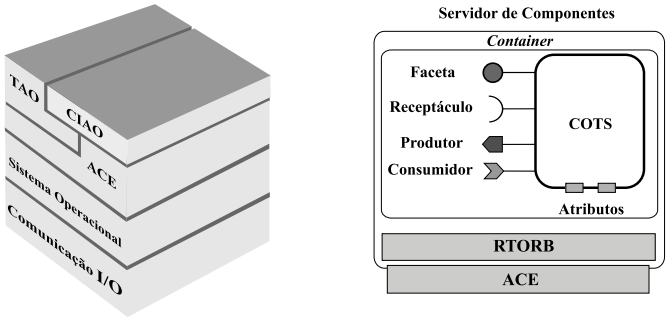
O trabalho desenvolvido considera um sistema composto por componentes que se comunicam através da conexão de *interfaces* padronizadas, conhecidas como *portas*. Através das portas, o componente especifica quais as funcionalidades que foram programadas, e que serão disponibilizadas externamente como serviços a outros componentes. Desse modo cada componente pode prover ou consumir serviços através de cada conexão que é estabelecida entre portas denominadas de *provedoras* e *receptoras*. Cada componente é executado por um processo, que se comunica por intermédio da troca de mensagens nos canais de comunicação. É assumido que

estes canais suportam restrições temporais e permitem comunicação síncrona e assíncrona. A perda de *deadlines* é admitida apenas em níveis toleráveis pelo sistema (*soft real-time*).

3.2. *Middleware* de Tempo-Real

Como o estudo de caso foi realizado no *middleware* CIAO (*Component-Integrated ACE ORB*)[Wang et al. 2004a], usamos seu modelo e terminologia ao longo do texto. No entanto, o método proposto é genérico o suficiente para ser aplicado a outros modelos de componentes. O modelo do CIAO está em conformidade com a especificação CCM (*Corba Component Model*)[OMG 2006]. Na terminologia adotada pelo CIAO as portas provedoras são denominadas Facetas, e as portas consumidoras são denominadas de Receptáculos. Para a comunicação assícrona, os canais de eventos são denominados de Produtores e Consumidores de eventos. As características internas de um componente podem ser configuradas através dos Atributos.

O CIAO foi desenvolvido sobre o TAO (The ACE ORB)[Schmidt et al. 1998], um *middleware* que disponibiliza um ORB (*Object Request Broker*) codificado em C++, e que adere à especificação Corba de tempo real. A estrutura interna de código do TAO tomou por base o ACE (*Adaptative Communication Environment*), um *framework* orientado a objetos que aplica padrões de projeto para comunicação de *software* que envolva distribuição e concorrência [Schmidt and Huston 2003]. Os serviços disponibilizados no TAO (e.g. serviço de escalonamento e serviço de eventos de tempo-real) foram projetados usando os padrões disponíveis no ACE, e tendo em vista a otimização da eficiência e previsibilidade para construir um ORB de tempo real que pudesse garantir os requisitos de QoS fim-a-fim para diversas aplicações [Schmidt and Cleeland 2001]. Com isso, o CIAO passa a contar com o suporte necessário para o desenvolvimento de soluções previsíveis, portáveis e interoperáveis. Uma visão geral da arquitetura utilizada é ilustrada na figura 1(a). Maiores detalhes técnicos internos da estrutura apresentada podem ser obtidos em [Wang et al. 2004b].



(a) Arquitetura (b) Servidor de Componentes

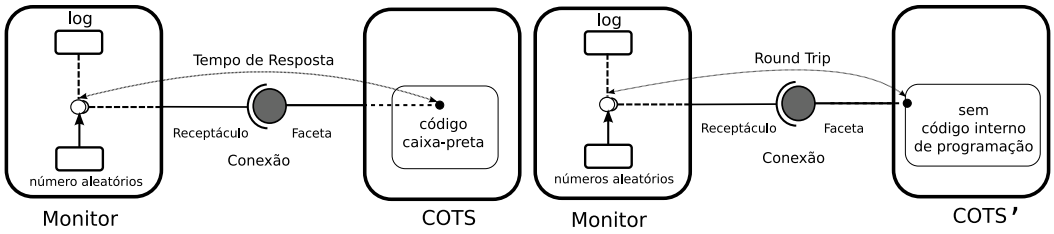
Figura 1. Middleware de tempo real baseado em componentes

O CIAO estende a especificação CCM ao desenvolver um servidor de componentes de tempo real, que inclui um ambiente de execução gerenciável capaz de prover suporte à execução e comunicação previsível de componentes, através da oferta de serviços não funcionais que podem ser configuráveis por intermédio de

descritores de implantação (arquivos XML - *eXtensible Markup Language*). Esse ambiente é chamado de *container*. Os serviços disponibilizados pelo *container* são providos através de *interfaces*, que permitem a configuração de políticas de QoS, além da interação com serviços relacionados a pesquisa, gerenciamento de conexões e ciclo de vida dos componentes instalados no servidor. O CIAO disponibiliza o DAnCE [Deng et al. 2005], uma solução para a automatização de procedimentos de implantação e configuração de componentes. Adicionalmente, o DAnCE oferece o serviço de reconfiguração dinâmica para componentes já implantados (ReDaC). Na figura 1(b) é ilustrado o servidor de componentes de tempo real presente no CIAO, e os conectores que representam a terminologia usada.

4. Estimando uma Distribuição de Probabilidade

O método proposto é baseado em medição, e requer a análise do tempo de resposta(R) e do tempo de comunicação entre componentes, como informações necessárias no estudo do tempo de execução de um serviço disponibilizado por um componente. Para viabilizar esta análise, o modelo de medição ilustrado na figura 2 estabelece que um componente *monitor* é responsável por medir a variável R associada à execução de um serviço. O *monitor* se encarrega também de medir o tempo de comunicação associado à execução de um serviço *nulo* (i.e., sem código interno de programação da *interface* publicada). Neste trabalho, este tempo foi denominado de *round-trip*(RT). Então, as variáveis R e RT são definidas como as variáveis de interesse. É importante ressaltar que o modelo de medição não requer qualquer conhecimento sobre o código interno da aplicação utilizada durante as medições. Neste caso, o código da aplicação é visto como uma entidade caixa-preta.



(a) Medição de R (b) Medição de RT

Figura 2. Modelo de medição adotado

O objetivo principal da abordagem proposta é estimar a distribuição de probabilidade do tempo de execução de um serviço. Neste sentido, o tempo de execução pode ser analisado através do tempo de resposta observado em um número grande o suficiente de requisições a um serviço. Deste modo, os valores da variável tempo de execução(C) podem ser estimados através da equação $C = R - RT$. Para viabilizar isso, o componente *monitor* é projetado para realizar diversas medições de valores de R e RT. Antes de detalharmos a abordagem, algumas definições são introduzidas a seguir.

Sejam S_r e S_{rt} duas coleções de valores medidos, que contém os tempos de reposta e de *round-trip* obtidos pelo *monitor*, respectivamente. Os elementos de cada coleção não são necessariamente distintos, e o conjunto de valores distintos em

uma coleção S é dados por $D(S)$. O número de vezes que um valor v é observado em uma coleção S é definido por $\eta(v, S)$. A função $f(v, S)$ fornece a frequência relativa do valor v na coleção S. Por exemplo, se o *monitor* mediu três vezes o mesmo valor $r \in S_r$ em 10 medições realizadas, então $f(r, S_r) = 0.3$. Mais precisamente:

$$f(v, S) = \frac{\eta(v, S)}{\sum_{\forall u \in D(S)} \eta(u, S)} \tag{1}$$

É possível derivar a distribuição de probabilidade de C calculando a distribuição de probabilidade conjunta:

$$P(C = c) = \sum P(R = r, RT = rt) \qquad \forall r \in S_r, rt \in S_{rt} : c = r - rt \tag{2}$$

Contudo, nem todas as combinações de valores em $r \in S_r$ and $rt \in S_{rt}$ são possíveis uma vez que $P(C \leq 0) = 0$. De forma geral, assumindo que deve existir um limite inferior $c_{min} > 0$ para o valor de C, é necessário considerar aquelas combinações que satisfaçam $r - rt \geq c_{min}$. É importante destacar que valores inválidos de $r - rt$ podem aparecer dado que as medições de S_r e S_{rt} são realizadas de forma independente. Então, é importante definir a coleção de valores possíveis para C como:

$$S_c = \{r - rt | r - rt \geq c_{min}, r \in S_r, rt \in S_{rt}\} \tag{3}$$

Assumindo $c_{min} \approx 0$ é simples, mas pouco realista. Logo, é conveniente usar uma estimativa mais apropriada de c_{min}, de forma que seja possível prover melhores resultados de aproximação quando se estiver derivando a distribuição de probabilidade de C. Uma vez que valores altos de tempo de execução são de maior interesse, o seguinte procedimento pode ser usado para estimar c_{min}:

1. Encontrar o menor valor $rt_u \in S_{rt}$ de modo que $P(RT \leq rt_u) \geq p$, e p é um parâmetro mínimo que define a probabilidade desejada pelo usuário $(0 < p < 1)$, e pode ser calculada como:

$$P(RT \leq rt_u) = \sum_{\forall rt \in D(S_{rt}) : rt \leq rt_u} f(rt, S_{rt}) \tag{4}$$

2. Encontrar o menor valor $r_{min} \in S_r$ de modo que $r_{min} - rt_u > 0$.
3. Definir $c_{min} = r_{min} - rt_u$.

Uma vez que c_{min} é encontrado, a coleção S_c e a função (1) produzem a distribuição de C. Para ilustrar a abordagem de estimativa proposta, vamos definir duas coleções $S_r = \{1, 2, 3, 6, 6, 7\}$ and $S_{rt} = \{1, 2, 3, 3, 3, 4\}$, e considerar o parâmetro mínimo desejado $p = 0.8$. Neste caso, $rt_u = 3$ visto que $P(RT \leq 2) = 1/3$ e $P(RT \leq 3) = 5/6 \approx 0.83$. Consequentemente, para $r_{min} = 6$ o valor de C não pode ser menor do que $c_{min} = 3$. Então, $S_c = \{3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5, 5, 6\}$.

É importante enfatizar que a definição acima de rt_u é usada para derivar o limite inferior c_{min}. O descarte de valores de C abaixo deste limite é útil uma vez

que isto melhora a qualidade dos resultados, e não compromete o procedimento de estimativa. De fato, a quantidade de valores descartados é muito pequena e apenas observada para valores de *round-trip* muito altos e muito pouco prováveis em relação ao r_{min}. Uma vez que é de maior interesse estimar melhor valores mais altos de C, esta estratégia de descarte é também uma abordagem segura. A próxima seção relata um estudo de caso que ilustra a eficiência do método de estimativa proposto.

5. Estudo de Caso e Avaliação

O método proposto foi aplicado em um experimento de teste realizado sobre o CIAO, um *middleware* baseado em componentes descrito na subseção 3.2. A configuração padrão do CIAO foi utilizada como base durante o procedimento de instalação, devido ao fato de a mesma já possuir característica destinadas a otimização e eficiência da plataforma, propiciando um ambiente adequado a muitas aplicações de tempo real. Contudo, dada a finalidade de medição do experimento, alguns parâmetros do ORB foram especificamente configurados através de arquivos descritores XML: política de escalonamento FIFO, mapeamento direto de prioridades CORBA para as prioridades nativas, políticas de prioridades definidas pelo servidor e uso de tarefas estáticas para a execução dos componentes instalados no *container*. Tais parâmetros implicam que a interferência nas medições será limitada, pois estes ajustes na configuração reduzem o *overhead*, o número de tarefas concorrentes e mantém fixa a prioridade durante o experimento, além de garantir a reserva de recursos no servidor de componentes (e.g. alocação prévia de memória).

O CIAO 0.6.0 foi compilado com o GCC 4.2.1 e instalado sobre o sistema operacional Linux com kernel 2.6.20-16, sendo utilizado um *hardware* com processador Intel Centrino Duo Core de 1.83GHz, 1GB de RAM e 2MB de memória *cache*. O sistema operacional foi configurado com os parâmetros `maxcpus` = 1, de modo a considerar apenas um núcleo de processador, e `runlevel` = 1, restringindo o sistema operacional aos seus serviços básicos. As definições dos serviços publicados e consumidos pelos componentes do experimento foram feitas através da linguagem IDL (*Interface Defination Language*), que permite a especificação e posterior geração da estrutura de componentes na arquitetura do CIAO. A categoria de compenentes escolhida foi a baseada em Sessão. O procedimento de instalação de componentes no *container* foi realizado com o auxílio do DAnCE, que seguiu um plano de instalação das composições de componentes utlizados no experimento. Este plano também foi baseado em descritores XML.

Em uma primeira fase, destinada a medir os valores da variável R, dois componentes foram conectados e instalados no *container* do servidor de componentes: um componente *monitor* e um componente a ser monitorado. Este último componente disponibiliza um serviço que simula um analisador de dados, que mantém registros de amostras de dados e provê análises a respeito do comportamento dos dados avaliados. Uma vez que a semântica desse serviço não é relevante aqui, nós não iremos tratá-lo em maiores detalhes e será referenciado apenas como serviço *analisador*. O componente *monitor* foi projetado e desenvolvido segundo alguns critérios específicos:

1. Permitir a configuração do mecanismo gerador de dados aleatórios utilizados como parâmetros de entrada para a execução do serviço monitorado. Isso

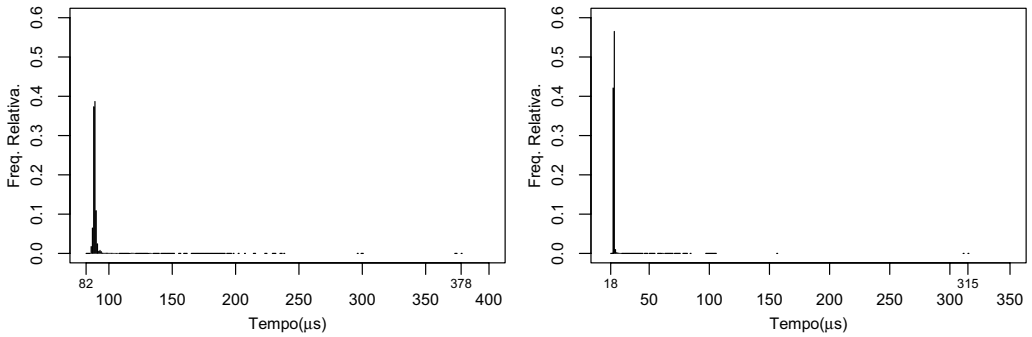
permite o ajuste e consequente redução da quantidade de estados a que o componente monitorado estará sendo observado, e produz especial impacto sobre a carga de processamento necessária durante a análise dos dados estatísticos. O ajuste adequado deve ser definido pelo projetista de acordo com os interesses do projeto.

2. A execução de um serviço deve ser medida ao menos 100 vezes para cada parâmetro de entrada fornecido. Isso permitirá a captura de um conjunto de valores de tempo obtido em diferentes estados do ambiente de execução. Desse modo as interferências da memória *cache* ou do *pipeline* podem ser observadas indiretamente nos tempos medidos.

Em uma segunda fase, destinada a medir os valores da variável RT, um terceiro componente foi conectado ao *monitor* e instalado no mesmo *container*. Contudo, este novo componente disponibiliza apenas um serviço *nulo*, e sua finalidade serve apenas para estimar valores de *round-trip* entre componentes (e.g. figura 2(b)). Para minimizar a interferência nos tempos medidos as prioridades de execução dos componentes monitorados foram fixadas no máximo valor permitido pelo Linux, e cada componente foi executado por uma tarefa dedicada.

Para realizar as medições, o *monitor* fez n chamadas através da conexão com os serviços *analisador* e *nulo*. Para cada chamada completada, o tempo medido foi registrado. Ambos os serviços utilizaram uma faceta idêntica, de forma que representassem uma mesma *interface*, e os parâmetros de chamada foram gerados aleatoriamente pelo *monitor* de acordo com uma faixa de valores de interesse. As medições realizadas não foram correlacionadas, e os valores de tempos das variáveis R e RT foram medidos de acordo com as frequências de chamada pré-estabelecidas na configuração do *monitor*: 25MHz, 50MHz, 75MHz e 100MHz. Então, foram realizadas $n/4$ medições para cada frequência de chamada. Vale ressaltar que valores distintos de frequência possibilitam medições em diferentes estados do ambiente de execução (*cache* e *pipeline*). As frequências citadas foram escolhidas por serem representativas em termos de sistemas de tempo real. Por outro lado, testes preliminares demonstraram que a partir de 800MHz o impacto causado por trocas de contexto no sistema operacional começa a gerar interferências significativas, que implicam em distorções nos tempos medidos. Para viabilizar o mecanismo de medição do *monitor*, nós utilizamos a classe *High Resolution Timer* provida pelo *framework* ACE, de modo que os tempos foram medidos na escala de microsegundos (μs). A interferência gerada por este mecanismo foi de aproximadamente 20 nanosegundos (ns), o que foi considerado como um impacto mínimo e desprezível. O número n de medições pode ser definido pelo usuário, e deve ser um valor alto o bastante para que os valores de tempo coletados durante as medições sejam representativos. Em nosso experimento, o valor utilizado foi $n = 2 \times 10^6$.

A figura 3 exibe a distribuição de probabilidade derivada a partir dos valores coletados em S_r e S_{rt}. Como pode ser visto, estas distribuições exibem um comportamento similar, embora a dispersão de S_r seja maior. Isto ocorre devido ao fato de os valores de S_r estarem mais sujeitos a maiores interferências quando comparados aos valores de S_{rt}. A dispersão pode ser considerada como esperada dado que não se está usando um sistema oparacional de tempo real [Liu 2000, Regnier et al. 2008].



(a) Tempo de Resposta (b) Round-Trip

Figura 3. Distribuição de probabilidade de R e RT.

Tabela 1. Análise Descritiva.

	R	RT	C	C'
Moda	89	21	68	66
Mediana	89	21	68	66
Média	88.8	20.62	68.20	65.87
Desvio padrão	2.317528	1.143541	2.381033	1.699584
mínimo	82	18	60	61
máximo	378	315	360	352

Dadas as coleções S_r e S_{rt}, o método de estimativa proposto foi aplicado. A tabela 1 exibe algumas informações da análise descritiva referentes aos resultados obtidos. Para avaliar o método proposto, nós também realizamos as medições do tempo de execução do componente que disponibilizou o serviço *analisador*. Logo, foi necessário instrumentar o código interno deste serviço. Os tempos de execução medidos foram representados pela variável C' na tabela. Note que também existe um desvio padrão mais alto de R quando comparado com RT. Isto é causado pela já mencionada dispersão de S_r quando comparada com S_{rt}.

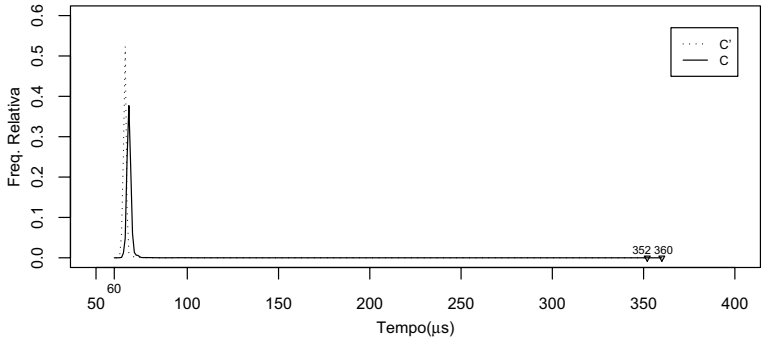


Figura 4. Distribuição das C e C'

Para a estimativa do c_{min} foi adotado $p = 99,5\%$, e o menor valor de rt_u que atendeu a esse requisito foi 22 unidades de tempo com probabilidade $99,59\%$. A partir desse valor as distribuições foram calculadas. As figuras 4 e 5 ilustram

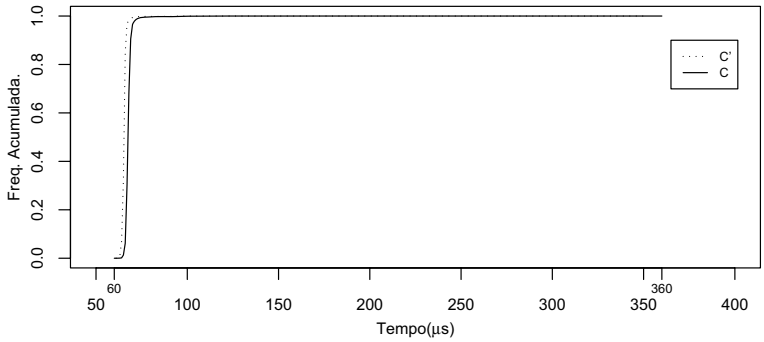


Figura 5. Distribuição acumulada das variáveis C e C'.

melhor a eficiência da abordagem proposta. Como pode ser visto, as distribuições de probabilidade de C, C' e suas respectivas funções de massa são muito similares. Na figura 4 a distribuição de C exibe uma curva deslocada para a direita, em relação à curva da distribuição de C'. Isto ocorre devido a maior influência causada pelo cálculo da probabilidade conjunta que envolve a *moda*, quando a equação (2) foi aplicada - i.e. $P\left(r = 89 \wedge rt = 21\right) \cong 21.86\%$. Contudo, este deslocamento à direita não compromete os resultados obtidos, visto que as probabilidade estimadas na distribuição de C são relativas a valores de tempo que são maiores do que os valores de tempo relacionados com a curva de probabilidade da distribuição C'. Portanto, a abordagem é segura quanto à adoção do valor estimado. O mesmo deslocamento à direita ocorre com relação à distribuição acumulada de C, o que é ilustrado na figura 5. De fato, $P\left(C \leq 100\right) = 99.93\%$ e $P\left(C' \leq 100\right) = 99.99\%$.

Finalmente, é possível notar que a distribuição de probabilidade estimada provê um meio simples de se obter um valor de WCET. Por exemplo, o valor $360\mu s$ com uma garantia probabilística $P\left(C < 360\right) \cong 99.999\%$. Outra possibilidade é utilizar essa distribuição em modelos probabilísticos de análise de desempenho como em [Kim et al. 2005] ou em [Manolache et al. 2001]. Nestes trabalhos a taxa de perda de *deadlines* é calculada por tarefa e a análise usa uma distribuição pseudo-contínua, que é baseada na discretização da curva de densidade de probabilidade.

6. Conclusões e Trabalhos Futuros

Este artigo apresentou e avaliou uma metodologia para a estimativa de uma distribuição de probabilidade de tempos de execução de componentes, voltada para sistemas de tempo real baseados em componentes COTS. Tais estimativas probabilísticas estão sendo consideradas cada vez mais relevantes no projeto de sistemas de tempo real modernos dado a dificuldade presente no uso de técnicas tradicionais de derivação do pior tempo de execução. A metodologia proposta é do tipo caixa-preta, não necessitando acesso ao código dos componentes, o que a torna altamente portável. A metodologia foi avaliada através de um estudo de caso, a partir do qual observou-se grande aproximação entre as distribuições de probabilidade estimada e medida, demonstrando assim a adequação da proposta aqui descrita.

Embora os experimentos aqui realizados não considerem uma composição de componentes, acreditamos que a metodologia proposta possa ser estendida para

lidar com tais cenários. Por exemplo, a partir de resultados obtidos com a execução de componentes isolados pode-se realizar convoluções que representem composições de componentes. Um tratamento estatístico adequado para derivar tais convoluções deve ser usado. Outra estratégia é realizar medições que considerem o tempo de resposta de composições. Para tanto, um sistema operacional de tempo real deve ser usado para que as interferências nas medições possam ser mantidas sob controle e incorporadas na análise. Tais aspectos devem ser considerados em trabalhos futuros.

Para validar a metodologia proposta em sistemas mais complexos, já se encontra em fase inicial os trabalhos para implantação do modelo de medição no ARCOS (ARquitetura para COntrole e Supervisão) [Andrade and Macêdo 2007], um *framework* baseado em componentes que permite o desenvolvimento de sistemas industriais modernos e distribuídos para a realização de tarefas de controle e supervisão industriais. O ARCOS utiliza o DAIS (*Data Acquisition from Industrial Systems*) [OMG 2005], um padrão criado pela OMG (*Object Management Group*) para a aquisição de dados industriais, que permite a transferência de uma grande quantidade de dados industriais simultaneamente para vários clientes (consumidores). Os objetivos futuros para o ARCOS consistem em utilizar a proposta de estimativa de distribuição de probabilidade de C nos testes de viabilidade de escalonamento dos diferentes serviços supervisionados, e estabelecer análises estatísticas dos tempos de resposta destes serviços.

Referências

Andrade, S. and Macêdo, R. (2007). Engineering components for flexible and interoperable real-time distributed supervision and control systems. In *12th IEEE Conference on Emerging Technologies and Factory Automation*, Patras - Greece.

Audsley, N., Burns, A., Richardson, M., and Tindell, K. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292.

Ballabriga, C., Cassé, H., and Sainrat, P. (2007). Wcet computation on software components by partial static analysis. Junior Researcher Workshop on Real-Time Computing. IRIT - Université de Toulouse, France.

Bernat, G., Colin, A., and Petters, S. M. (2002). Wcet analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium*, pages 279– 288, Washington. IEEE Computer Society.

Deng, G., Balasubramanian, J., Otte, W., Schmidt, D. C., and Gokhale, A. (2005). Dance: A qos-enabled component deployment and configuration engine. In *Proceedings of the 3rd Working Conference on Component Deployment*, Grenoble, France.

Edgar, S. and Burns, A. (2001). Statistical analysis of wcet for scheduling. In *22nd IEEE Real-Time Systems Symposium*, page 215, Washington, DC, USA. IEEE Computer Society.

Estévez-Ayres, I., García-Valls, M., and Basanta-Val, P. (2005). Enabling wcet-based composition of service-based real-time applications. *SIGBED Review*, 2:25–29.

Fredriksson, J. (2006). Increasing accuracy of property predictions for embedded real-time components. In *18th Euromicro Conference on Real-Time Systems*.

Kim, K., Diaz, J. L., Lopez, J. M., Bello, L. L., Lee, C.-G., and Min, S. L. (2005). An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Trans. Comput.*, 54(11):1460–1466.

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61.

Liu, J. W. S. (2000). *Real-Time Systems*. Prentice Hall, 1 edition.

Manolache, S., Eles, P., and Peng, Z. (2001). Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In *13th Euromicro Conference on Real-Time Systems*, page 19, Washington, DC, USA. IEEE Computer Society.

Möller, A., Peake, I., Nolin, M., Fredriksson, J., and Schmidt, H. (2005). Component-based context-dependent hybrid property prediction. In *Workshop on Dependable Software Intensive Embedded systems*.

Moss, A. and Muller, H. (2004). Model generation for temporal properties of reactive components. In *1st International Workshop on Software Analysis and Development for Pervasive Systems*, pages 12–19.

Nolte, T., Möller, A., and Nolin, M. (2003). Using components to facilitate stochastic schedulability analysis. In *24th IEEE Real-Time System Symposium - WiP Session*, Cancun, Mexico.

OMG (2005). *Data Acquisition from Industrial Systems v1.1*. formal/2005-06-01.

OMG (2006). *CORBA Component Model Specification*.

Perrone, R., Macêdo, R., Lima, G., and Lima, V. (2008). Estimating execution time probability distributions in component-based real-time systems. In *Proceedings of the 10th Brazilian Workshop on Real Time and Embedded Systems*, Rio de Janeiro - RJ, Brazil. Brazilian Computer Society.

Regnier, P., Lima, G., and Barreto, L. P. (2008). Avaliação do Determinismo Temporal no Tratamento de Interrupções em Plataformas de Tempo Real Linux. In *Proc. of the Brazilian Workshop on Operating Systems (WSO 08)*. Brazilian Computer Society.

Schmidt, D. C. and Cleeland, C. (2001). Applying a pattern language to develop extensible orb middleware. pages 393–438.

Schmidt, D. C. and Huston, S. D. (2003). *C++ Network Programming: Systematic Reuse with ACE and Frameworks*. Addison-Wesley Longman.

Schmidt, D. C., Levine, D. L., and Mungee, S. (1998). The design of the tao real-time object request broker. *Computer Communications*, 21(4):294–324.

Wang, N., Gill, C., Subramonian, V., and Schmidt, D. C. (2004a). Configuring real-time aspects in component middleware. *Proceedings of the International Symposium on Distributed Objects and Application*, pages 1520–1537.

Wang, N., Gill, C. D., Schmidt, D. C., Gokhale, A., Natarajan, B., Loyall, J. P., Schantz, R. E., and Rodrigues, C. (2004b). *Middleware for communications*, chapter QoS-enabled Middleware, pages 131–159. John Wiley & Sons.