Predição de Falhas em *Workflows* Científicos com o uso de Redes de Petri Estocásticas e Lógica $\mathcal{DS}_3$

Bruno Lopes[1], **Daniel de Oliveira**[1]

[1]Instituto de Computação — Universidade Federal Fluminense (UFF)

{bruno,danielcmo}@ic.uff.br

Abstract. *Several existing workflows require many computational resources because they process a large amount of data. Thus, High Performance Processing (PAD) environments must be applied in conjunction with parallelization techniques to support their execution. Although PAD environments offer several advantages, failures are a reality, not a possibility, due to the large number of computing nodes involved in the execution of the workflow. Verifying workflow failures is a challenge that still requires effort. In this paper we propose the use of $\mathcal{DS}_3$, a dynamic logic taiored to reason about stochastic Petri nets, to verify and predict failures in workflows.*

Resumo. *Vários workflows existentes exigem muitos recursos computacionais, pois processam um grande volume de dados. Dessa forma, os ambientes de Processamento de Alto Desempenho (PAD) devem ser aplicados em conjunto com técnicas de paralelização para apoiar a sua execução. Embora os ambientes de PAD ofereçam diversas vantagens, as falhas são uma realidade, e não uma possibilidade, devido ao grande número de nós de computação envolvidos na execução do workflow. Verificar falhas no workflow é uma tarefa desafiadora que ainda requer esforço. Neste artigo, propomos o uso da $\mathcal{DS}_3$, uma lógica dinâmica adaptada sobre as redes de Petri estocásticas, para verificar e prever falhas em workflows.*

1. Introdução

Os *Workflows* científicos (ou simplesmente *workflows*) podem ser definidos como uma especificação formal de um processo científico, que representa as etapas a serem executadas dentro de um experimento científico *in silico* [Deelman et al. 2009]. Tais etapas (*i.e.,* atividades) são comumente associadas a invocações de programas e/ou serviços que realizam uma série de transformações sobre dados científicos, *i.e.,* seleções de dados, agregação de dados, filtragem e sumarização, análise e visualização. Um *workflow* pode ser formalmente definido como um grafo acíclico dirigido $Wf(A, Dep)$, onde os nós $A = \{a_1, a_2, ..., a_n\}$ são as atividades e as arestas Dep representam as dependências de dados entre atividades em A. Assim, dado $a_i \mid (1 \leq i \leq n)$, o conjunto $P = \{p_1, p_2, .., p_m\}$ representa os possíveis parâmetros de entrada (*e.g.*, valores, ponteiros de arquivo, *etc*) para a atividade a_i que define o comportamento de a_i.

Definamos uma *ativação* [Ogasawara et al. 2011] como a menor unidade de trabalho que pode ser processada em paralelo e consome um *chunk* de dados específico [Liu et al. 2015]. Vamos considerar $Ac = \{ac_1, ac_2, ..., ac_k\}$ como o conjunto de ativações do *workflow* Wf. Cada ac_i está associada a uma atividade específica a_j que é representada como $act(ac_i) = a_j$. As ativações também apresentam dependências de dados, portanto, $input(ac_i) \in I$ e $output(ac_i) \in O$ e a dependência entre duas ativações ac_i e ac_j pode ser representada como $dep(ac_i, ac_j) \leftrightarrow \exists r \in input(ac_j) | r \in output(ac_i) \wedge dep(act(ac_i), act(ac_j))$. Definamos $PV_i = \{pv_{1i}, pv_{2i}, .., pv_{mi}\}$ como os valores de parâmetros consumidos por uma ativação ac_i.

Os *workflows* são geralmente modelados, executados e monitorados por mecanismos complexos chamados Sistemas de Gerência de *Workflows* (SGWf) que apoiam a especificação em termos de artefatos executáveis. De acordo com [Liu et al. 2015], SGWfs bem conhecidos são o Pegasus, Swift/T, e o SciCumulus. A maioria desses SGWfs, além da execução de ativações, captura dados históricos do *workflow*, chamados de dados de proveniência [Freire et al. 2008].

Um *workflow* pode ser executado quantas vezes forem necessárias no contexto de um experimento científico. Os cientistas geralmente variam parâmetros de entrada e dados para avaliar uma hipótese científica. Em muitos experimentos, os cientistas não sabem *a priori* quais valores e parâmetros produzem os melhores resultados, então eles têm que explorar uma faixa de valores, similar a um processo de tentativa e erro. O problema dessa análise de parâmetros por meio de tentativa e erro é que ela pode definir algumas combinações de parâmetros que incorrem em erros de execução ou produzem resultados não úteis.

Esse fato se torna ainda mais complexo se considerarmos que vários *workflows* existentes são executados em paralelo usando ambientes de processamento de alto desempenho (PAD) que apresentam custos associados, como nuvens públicas (*e.g.,* Amazon AWS e MS Azure) ou redes comunitárias[1]. Em uma execução paralela do *workflow*, cada ativação ac_i pode ser escalonada para muitos nós computacionais (CNs) no ambiente $CN = \{cn_1, cn_2, ..., cn_d\}$ e cada cn_d apresenta diferentes capacidades de processamento, memória e disco. Neste caso, a presença de falhas pode gerar, além de atrasos na execução do *workflow* (que já são indesejáveis), questões financeiras. Portanto, é interessante que a execução do *workflow* deva ser minimamente impactada por falhas na execução de suas ativações.

No entanto, está longe de ser trivial identificar possíveis falhas em uma execução de *workflow*, especialmente quando o mesmo é composto de várias ativações executadas em paralelo em um ambiente de PAD. Os dados de proveniência podem desempenhar um papel importante nessa tarefa de previsão de falhas, pois podem fornecer informações históricas que podem ser usadas para identificar padrões comuns de falha ou para calcular a probabilidade de uma falha ocorrer. No entanto, ainda é uma tarefa árdua identificar tais padrões que podem ser usados pelos SGWfs existentes para prever falhas e informar ao cientista que uma determinada ativação ac_i consumindo determinado conjunto de valores de parâmetro PV_i provavelmente gerará erros de execução ou produzirá resultados indesejados.

Neste artigo, propomos o uso da $\mathcal{DS}_3$ [Lopes et al. 2014], uma lógica dinâmica adaptada para raciocinar sobre redes de Petri Estocásticas (SPN) [Marsan 1990, Marsan and Chiola 1987], para verificar e prever falhas nos *workflows* com base nos dados de proveniência previamente coletados. É bem conhecido que os *workflows* se beneficiam das redes de Petri para verificar as execuções que levam ao resultado desejado, independentemente de como ele pode ser executado. A vantagem de usar a $\mathcal{DS}_3$ em comparação com a abordagem tradicional de redes de Petri para prever falhas em *workflows* é que o poder expressivo é aumentado combinando-o com modelos lógicos e tirando proveito de uma abordagem estocástica especialmente ao lidar com casos de altos custos computacionais. Avaliamos nossa abordagem usando *traces* do *workflow* Montage executado em um *cluster*.

Este artigo está organizado da seguinte maneira. A Seção 2 apresenta um referencial sobre redes de Petri Estocásticas (SPN) e sobre a lógica $\mathcal{DS}_3$. A Seção 4 apresenta o *design* do sistema proposto, *i.e.,* como a lógica $\mathcal{DS}_3$ e o SPN podem ser acoplados a SGWf existentes. A Seção 5 apresenta a aplicação da abordagem como uma prova de conceito. A Seção 6 traz os

[1]https://internuvem.usp.br/client/

trabalhos relacionados. Finalmente, a Seção 7 conclui este artigo e aponta trabalhos futuros.

2. Redes de Petri Estocásticas e a Lógica $\mathcal{DS}_3$

As redes de Petri Estocásticas (SPN) são amplamente usadas para modelar e raciocinar sobre sistemas dinâmicos, que é o caso da execução paralela de *workflows* com PAD. Uma rede de Petri estocástica é uma tupla $\mathcal{P} = \langle S, T, W, O_0, \Lambda \rangle$, onde (i) S é um conjunto finito de lugares; (ii) T é um conjunto finito de transições, de tal forma que para cada uma associamos uma variável aleatória que segue a distribuição exponencial; (iii) $S \cap T = \emptyset$; (iv) $S \cup T \neq \emptyset$; (v) W é uma relação que define bordas direcionadas entre lugares e transições; (vi) O_0 é a relação de marcação inicial e (vii) $\Lambda = \langle \lambda_{t_1}, \ldots, \lambda_{t_n} \rangle$, $t_i \in T$ são as taxas de disparo de cada transição (*i.e.,* o parâmetro de cada variável aleatória distribuída exponencialmente associado a cada transição).

A marcação inicial (O_0) denota a distribuição inicial de *tokens* e escrevemos $O_i(s)$ para denotar a quantidade de *tokens* no local $s \in P$ na marcação O_i (a relação com a quantidade de fichas de cada lugar na marcação i). Pela predefinição de $t \in T$, denotada por ${}^\bullet t$, denotamos o conjunto de locais $S' \subseteq S$ tal que $\forall x \in S', (x, t) \in W$. Analogamente, o *postset*, denotado por $t^\bullet$ é o conjunto $S'' \subseteq S$ tal que $\forall x \in S'', (t, x) \in W$.

Por $\lambda(i)$ denotamos a relação com os atrasos de todas as transições. Então, dada uma marcação O_i de uma SPN, dizemos que t está habilitado em O_i se e somente se existir um *token* em cada lugar de sua pré-configuração, que é $\forall x \in {}^\bullet t, O_j(x) \geq 1$, e sua temporização é o mínimo de todos os $t_j \in T$, que é $\lambda_t(i) = \min(\lambda(i))$. A transição ativada pode ser acionada fornecendo uma nova marcação da seguinte maneira:

$$O_{i+1}(x) = \begin{cases} O_i(x) - 1, & \forall x \in {}^\bullet t \setminus t^\bullet \\ O_i(x) + 1, & \forall x \in t^\bullet \setminus {}^\bullet t \\ O_i(x), & \text{caso contrário} \end{cases} \tag{1}$$

O fluxo de atrasos é denotado como:

$$\lambda_{t_i}(j+1) \begin{cases} = & \text{new}_e(\lambda_{t_i}) \quad \text{se} \begin{cases} \begin{cases} \forall x \in {}^\bullet t_i, O_j(x) \geq 1 \\ \lambda_{t_i}(j) \leq \min(\lambda(j)) \end{cases} \\ \text{ou} \\ \begin{cases} \exists x \in {}^\bullet t_i, O_j(x) < 1 \\ \forall x \in {}^\bullet t_i, O_{j+1}(x) \geq 1 \end{cases} \end{cases} \\ < & \lambda_i(j) \qquad \text{caso contrário} \end{cases} \tag{2}$$

onde $\text{new}_e(\lambda_{t_i})$ denota uma nova ocorrência da variável aleatória distribuída exponencialmente com o parâmetro $\lambda_{t_i} \in \lambda$.

A lógica $\mathcal{DS}_3$ [Lopes et al. 2014] é uma lógica dinâmica na qual cada programa é uma rede de Petri Estocástica. Entre suas vantagens, temos que a $\mathcal{DS}_3$ é provada ser completa e decidível e possui alguns sistemas dedutivos. Recuperamos as definições originalmente apresentadas em [Lopes et al. 2014] que são necessárias neste artigo. A linguagem $\mathcal{DS}_3$ consiste em (i) Símbolos proposicionais: p, q ..., onde Φ é o conjunto de todos os símbolos proposicionais; (ii) Nomes de lugares: *e.g.,* $a, b, c, d \ldots$; (iii) Tipos de transição: $\mathbf{T_1} : at_1 b$, $\mathbf{T_2} : abt_2 c$ e $\mathbf{T_3} : at_3 bc$, cada transição tem um tipo exclusivo; (iv) Símbolo de composição da rede de petri: $\odot$ e (v) Sequência de nomes: $S = \{\epsilon, s_1, s_2, \ldots\}$, onde ϵ é a sequência vazia. Usamos a notação $s \prec s'$ para indicar que todos os nomes que ocorrem em s também ocorrem em s'.

Programa $\mathcal{DS}_3$. Um programa na lógica $\mathcal{DS}_3$ é um par (Π, Λ) em que Π é uma composição de transições definida como (seja s uma sequência de nomes - a marcação de Π):

(i) Programas básicos: $\pi_b ::= at_1 b \mid abt_2 c \mid at_3 bc$ onde t_i é do tipo $T_i, i = 1, 2, 3$; (ii) Programas de rede de Petri Estocásticas: $\pi ::= \pi_b \mid \pi \odot \pi$ e (iii) Programa de rede estocástica marcada de Petri: $\Pi = s, \pi$.

Fórmula na Lógica $\mathcal{DS}_3$. Uma fórmula é definida como (considere s uma sequência de nomes) $\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle s, \pi\rangle\varphi$. Usamos as seguintes abreviações padrão $\bot \equiv \neg\top$, $\varphi \vee \phi \equiv \neg(\neg\varphi \wedge \neg\phi)$, $\varphi \rightarrow \phi \equiv \neg(\varphi \wedge \neg\phi)$ e $[s, \pi]\varphi \equiv \neg\langle s, \pi\rangle\neg\varphi$.

Uma relação de disparo $f\colon S \times \Pi \rightarrow S$ é definida em [Lopes et al. 2014].

Frame $\mathcal{DS}_3$**.** Um Frame na lógica $\mathcal{DS}_3$ é uma tupla $\mathcal{F}_3 = \langle W, R_\pi, M, (\Pi, \Lambda), \delta\rangle$ onde (i) W é um conjunto não vazio de estados; (ii) $M\colon W \rightarrow S$; (iii) (Π, Λ) é uma rede de Petri estocástica onde Π é uma rede de Petri estocástica finita que para qualquer programa π usado em uma modalidade, $\pi \in \Pi$ (*i.e.,* π é uma sub-rede de Π). Além disso, $\Lambda(\pi) = \langle\lambda_1, \lambda_2, \ldots, \lambda_n\rangle$ é uma sequência de valores em $\mathbb{R}^+$ que denotam a taxa de disparo de cada transição de $\pi_1 \odot \pi_2 \odot \cdots \odot \pi_n = \pi \in \Pi$; (iv) $\delta(w, \pi) = \langle d_1, d_2, \ldots, d_n\rangle$ é uma sequência de retardos de disparo do programa $\pi \in \Pi$ em um mundo $w \in W$.

Modelo $\mathcal{DS}_3$**.** Um modelo $\mathcal{DS}_3$ é um par $\mathcal{M} = \langle\mathcal{F}_3, \mathbf{V}\rangle$, onde $\mathcal{F}_3$ é um frame $\mathcal{DS}_3$ e $\mathbf{V}$ é uma função de avaliação $\mathbf{V}\colon \Phi \rightarrow 2^W$.

Lema - Probabilidade de Veracidade de uma Modalidade A probabilidade de $\mathcal{M}_3$,$w \Vdash \langle s, \pi_b\rangle\varphi$ é (seja $s = M(w)$):

$$\Pr(\mathcal{M}_3,w \Vdash \langle s, \pi_b\rangle\varphi \mid \delta(w, \Pi)) = \frac{\delta(w, \pi_b)}{\sum_{\pi_b \in \Pi : f(s,\pi_b) \neq \epsilon} \delta(w, \pi_b)}$$

Noção Semântica da $\mathcal{DS}_3$**.** Seja $\mathcal{M}_3$ um modelo para $\mathcal{DS}_3$. A noção de satisfabilidade de uma fórmula φ in $\mathcal{M}_3$ em um estado w, dito $\mathcal{M}_3$,$w \Vdash \varphi$ é indutivamente definida da seguinte forma: (i) $\mathcal{M}_3$,$w \Vdash p$ sse $w \in \mathbf{V}(p)$; (ii) $\mathcal{M}_3$,$w \Vdash \top$; (iii) $\mathcal{M}_3$,$w \Vdash \neg\varphi$ sse $\mathcal{M}_3$,$w \nVdash \varphi$; (iv) $\mathcal{M}_3$,$w \Vdash \varphi_1 \wedge \varphi_2$ sse $\mathcal{M}_3$,$w \Vdash \varphi_1$ e $\mathcal{M}_3$,$w \Vdash \varphi_2$; (v) $\mathcal{M}_3$,$w \Vdash \langle s, \eta\rangle\varphi$ se existe $v \in W$, $wR_\eta v$ e $\Pr(\mathcal{M}_3,w \Vdash \langle s, \eta_\beta\rangle\varphi \mid \delta(v, \Pi)) > 0$ e $\mathcal{M}_3$,$v \Vdash \varphi$. Dessa maneira dizemos que $\mathcal{M}_3$,$w \Vdash \langle s, \eta\rangle\varphi$ então significa que o programa η que começa com a marcação s possui probabilidade de executar maior que um (*i.e.,* a probabilidade do disparo acontecer é maior que zero) e que quando para φ armazena o estado atual. Se φ é válida em todos os estados de $\mathcal{M}_3$ então φ é válida em $\mathcal{M}_3$, dizemos que $\mathcal{M}_3 \Vdash \varphi$; e se φ é válida em qualquer modelo então φ é válida, dizemos que $\Vdash \varphi$.

3. Abordagem Proposta

Nesta seção, apresentamos como acoplamos a lógica $\mathcal{DS}_3$ e a estrutura SPN com um SGWf existente. A lógica $\mathcal{DS}_3$ é implementada em um componente chamado FoWL (***F**ailure prediction in **W**orkflo**W** based on **L**ogic*). A arquitetura proposta é composta de 4 componentes principais (conforme apresentado na Figura 1): Componente ETL, FoWL, o SGWf e o banco de dados de proveniência.

O primeiro componente a ser invocado é o componente *ETL* que visa extrair informações do banco de dados de proveniência e a especificação do *workflow* a ser enviada para o componente *FoWL*. Vale a pena ressaltar que o componente *ETL* deve ser personalizado para cada tipo diferente de banco de dados de proveniência (*e.g.,* relacional, XML, RDF, grafo, *etc*). Em sua versão atual, implementamos um *script* python que extrai informações do banco de dados de proveniência e o representa no formato do *Workflow Generator* [Juve et al. 2013]. Uma vez

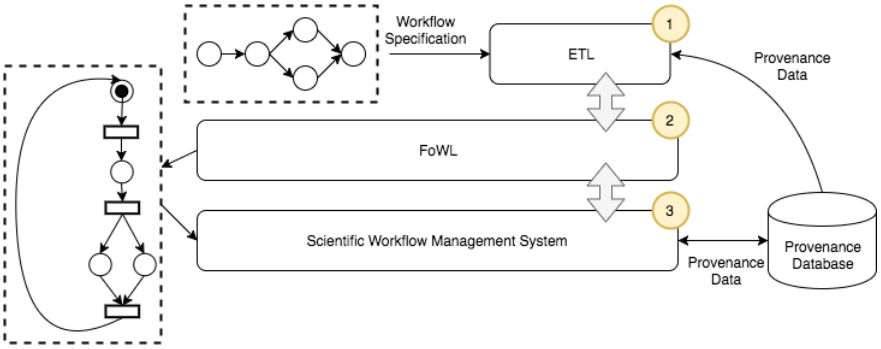


Figura 1. Arquitetura Abstrata da Abordagem Proposta

que o componente ETL converte as informações, ele invoca o componente FoWL que é um programa que usa a lógica $\mathcal{DS}_3$ para raciocinar sobre redes de Petri estocásticas, de forma a verificar e prever falhas de ativação nos *workflows* dados como entrada.

Uma vez identificadas as falhas (com uma probabilidade associada), o FoWL informa o SGWf para evitar a execução de tais ativações. Neste artigo, estendemos o SGWf SciCumulus para ler a lista de ativações que apresentam alta probabilidade de falha - maior que 60% de chance de falha. Como o SciCumulus é um SGWf orientado a banco de dados (todos os dados são armazenados em um banco de dados relacional), é simples informar ao SGWf quais são as ativações que apresentam alta probabilidade de falha usando um comando SQL UPDATE na tabela $eactivation$, *i.e.*, cada ativação é representada em uma tupla na tabela $eactivation$ e existe um campo que informa a probabilidade de falha fornecida pelo FoWL. Para mais informações sobre o esquema de proveniência do SciCumulus, consulte [de Oliveira et al. 2012].

4. Avaliação Experimental

Nesta seção, apresentamos a avaliação da abordagem proposta. Tomamos como estudo de caso o *workflow* Montage, pois é um benchmark *de fato* para avaliar abordagens de *workflows* científicos [Jacob et al. 2010].

4.1. Estudo de caso: o *Workflow* Montage

Nós modelamos o *workflow* Montage no SciCumulus usando o kit de ferramentas de astronomia Montage para montar imagens astronômicas em mosaicos personalizados usando um formato adequado para processamento de dados em grande escala de imagens do céu. Este kit de ferramentas compreende um conjunto de componentes que fornece serviços para construir mosaicos no formato de arquivo FITS (*Flexible Image Transport System*). O Montage usa diferentes imagens astronômicas para mesclá-las em mosaicos personalizados, considerando as transformações geométricas necessárias.

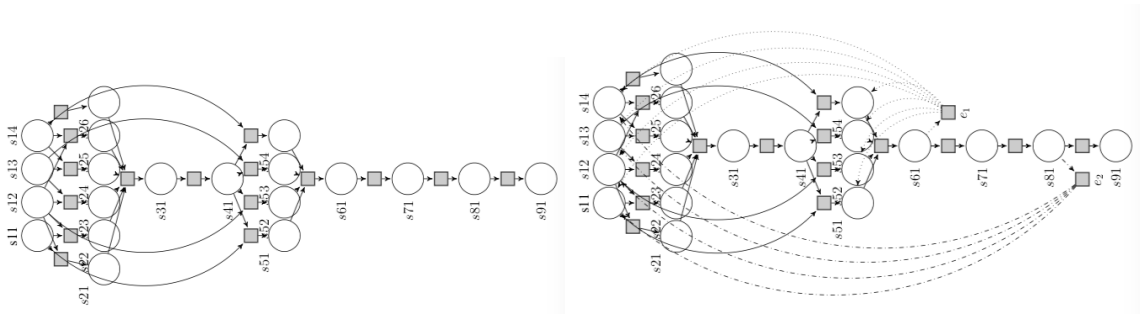
O Montage é composto por nove atividades. A primeira atividade (*List FITS*) extrai vários arquivos FITS de um arquivo compactado (obtido de um repositório externo de astronomia - 2MASS[2]) . A segunda atividade (Projeção) calcula a projeção dessas referências de posicionamento astronômico em um plano específico. As três atividades seguintes juntam os arquivos de projeção do FITS associados ao mesmo mosaico. As outras atividades do Montage são definidas para considerar interferências de sobreposição e correções de cores para criar um mosaico personalizado corrigido. Para mais detalhes sobre Montage, por favor consulte [Jacob et al. 2010].

[2]https://www.ipac.caltech.edu/2mass/releases/allsky/

4.2. O Montage Modelado como uma Rede de Petri Estocástica

A Figura 2(a) apresenta a tradução da especificação Montage para uma rede de Petri estocástica. Ela considera cada nó como um local e define uma transição para cada dependência no *workflow*. Usando um modelo $\mathcal{DS}_3$ pode-se verificar todas as propriedades padrão desejadas como a ausência de *deadlocks, liveness*, *etc*. É possível melhorar o modelo para tratamento de falhas. Na Figura 2(b) ampliamos a tradução básica para incluir a previsão de falhas.

Usando informações de proveniência de falhas passadas, podemos modelar a presença de falhas nas diversas ativações do *workflow*. Variáveis aleatórias distribuídas exponencialmente são amplamente usadas na literatura para modelar falhas. Usamos os dados de proveniência para identificar falhas de ativações e definir transições que representam a reação, *i.e.*, as ações que devem ser executadas quando ocorre uma falha (*e.g.,* e_1 e e_2); Depois disso, estimamos o parâmetro que por máxima verossimilhança $\hat{\lambda} = \frac{n}{\sum_1^n x_i}$.



(a) Rede de Petri Ordinária (b) Rede de Petri Estocástica

Figura 2. Implementação do Montage como uma Rede de Petri

Dessa forma, é possível prever falhas em ativações não apenas por simulação (como em redes de Petri padrão), mas também usando um modelo $\mathcal{DS}_3$ $\mathcal{M}_3 = \langle W, R_\pi, M, (\Pi, \Lambda), \mathbf{V} \rangle$. Seja $\pi \in \Pi$ a SPN correspondente ao programa e $e \in \Phi$ um símbolo proposicional que siginifica que a falha ocorreu quando a ativação $s61$ estava sendo executada. De forma a verificar se é possível que uma falha ocorra em $s61$ nós temos apenas que checar se a transição e_1 está ativada. Assumindo que sim, para checar a probabilidade desssa falha ocorrra no estado w, basta computar $\Pr(\mathcal{M}_3, w \Vdash \langle s, s61e_1p \rangle e \mid \delta(w, \Pi)) = \frac{\delta(w, s61e_1p)}{\sum_{s61e_1p \in \Pi : f(s, s61e_1p) \neq \epsilon} \delta(w, s61e_1p)}, s = M(w)$.

4.3. Resultados e Discussão

No experimento executado consideramos dados de proveniência de 100 execuções do Montage que podem ser obtidas no sítio do *Workflow Generator*[3](100 node DAX). Nestes dados inserimos falhas artificialmente de forma que tivéssemos ativações com falhas para que a $\mathcal{DS}_3$ e a FoWL pudessem detectar. A taxa de falha inserida foi de 20%. Falhas foram inseridas artificialmente nas ativações ID00003 (em 7 execuções do Montage) e ID00002 (em 3 execuções do Montage). As ativações ID00011, ID00019, ID00023, ID00025, ID00029, ID00033, ID00041 e ID00046 falharam devido a problemas de dependência de dados com a ativação ID00003. Por outro lado, as ativações ID00015, ID00018, ID00019, ID00032 e ID00040 falharam devido a problemas de dependência de dados com a ativação ID00002. Foram estimados os parâmetros das variáveis aleatórias para a rede de Petri da Figura 2 e definir as taxas de disparo.

[3]https://confluence.pegasus.isi.edu/display/pegasus/workflowgenerator

Para verificar se duas ativações quaisquer podem executar em paralelo (*e.g.,* de $s41$ até $s51$ e $s52$), nós temos que apenas verificar os resultados da função de disparo. Para verificar a probabilidade de uma execução de processar $s31$ com sucesso (seja π a rede de Petri completa), nós computamos $\Pr(\mathcal{M}, w \Vdash \langle M(w), s31t_1s41\rangle\top \mid \delta(w,\pi))$, que resulta em computar $\frac{\delta(w,s31t_1s41)}{\sum_{\pi_b\in\pi: f(M(w),\pi_b)\neq\epsilon} \delta(w,\pi_b)}$. Nesse experimento obtivemos o valor 0.95. Nós podemos também provar logicamente que o *workflow* irá falhar por completo (*e.g.,* $\mathcal{M} \Vdash \neg[M(w), s31t_1s41]\top$). A rede de Petri apresentada na Figura 2 tem duas transições de falha. A ideia é que o SGWf possa modificar a especificação do *workflow* ou o escalonamento do mesmo para evitar falhas a partir das informações geradas pela abordagem proposta nesse artigo.

5. Trabalhos Relacionados

A relação entre *workflows* e redes de Petri tem sido amplamente discutida como em [Salimifard and Wright 2001]. As redes de Petri podem ser usadas como uma linguagem de especificação para *workflows* complexos [van der Aalst 1998] com uma estrutura ampla para a verificação de correção da análise [van der Aalst 1998, Hofstede et al. 1998]. Seu uso leva à possibilidade de verificar propriedades conceituais como a ausência de *deadlocks, liveness*, *etc*. [Zhao et al. 2009] propõe um método de modelagem baseado no CCS para descrever comportamentos entre atividades em *workflows*. Eles adicionam restrições de dependência e parâmetros em expressões de ativações. Eles também fornecem uma linguagem de especificação para estabelecer um modelo formal. Sua abordagem é avaliada em um estudo de caso usando um protótipo desenvolvido.

Abordagens estocásticas como redes de Petri estocásticas também foram estudadas para análise de desempenho [Chuang et al. 2002]. São extensões de redes de Petri comuns onde é possível controlar o tempo dos eventos. Relacionar os *workflows* com a lógica também já está presente na literatura, como em [Curcin et al. 2009]. Estruturas lógicas são muito poderosas, mas a complexidade computacional da satisfatibilidade e prova de teoremas pode levar a cenários impraticáveis. [Liang and Zhao 2008] propõem um método de verificação para *workflows* que é baseado na lógica proposicional. Essa abordagem de verificação de *workflows* baseada em lógica apresenta algumas vantagens, como o formalismo lógico e capacidade de lidar com modelos genéricos de processos baseados em atividades. Eles mostraram que essa abordagem é capaz de detectar anomalias em *workflows*, mas não consideram probabilidades associadas.

6. Conclusões

Este artigo apresenta uma abordagem que combina lógica, um método formal e probabilidade de modelar e raciocinar sobre falhas em *workflows* computacionalmente intensivos. Mostramos como modelar *workflows* como redes de Petri e como usar a lógica $\mathcal{DS}_3$ para raciocinar sobre falhas de ativações. Usando o Montage definimos um experimento a partir da qual calculamos os parâmetros da SPN e apresentamos como calcular algumas propriedades desejadas, com foco na previsão de falhas, *i.e.,* podemos descobrir a probabilidade de uma ativação falhar a partir de processamento dos dados de proveniência.

Trata-se de um experimento como prova de conceito que indica que a abordagem vale à pena ser explorada. Espera-se que o uso de volumes maiores de dados gerem resultados mais representativos. Trabalhos futuros incluem experimentos mais sofisticados com mais instâncias do Montage e novos *workflows* e a integração com algumas ferramentas que automatizam o raciocínio.

Agradecimentos

A pesquisa apresentada nesse artigo foi parcialmente financiada pelo CNPq, CAPES e FAPERJ.

Referências

Chuang, L. I. N., Yang, Q. U., Fengyuan, R. E. N., and Marinescu, D. C. (2002). Performance equivalent analysis of workflow systems based on stochastic Petri net models. In *Engineering and Deployment of Cooperative Information Systems*, pages 64–79. Springer.

Curcin, V., Ghanem, M. M., and Guo, Y. (2009). Analysing scientific workflows with Computational Tree Logic. *Cluster Computing*, 12(4):399–418.

de Oliveira, D., Ocaña, K. A. C. S., Baião, F. A., and Mattoso, M. (2012). A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.*, 10(3):521–552.

Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, pages 20–30.

Hofstede, A. H. M., Orlowska, M. E., and Rajapakse, J. (1998). Verification problems in conceptual workflow specifications. *Data & Knowledge Engineering*, 24(3):239–256.

Jacob, J. C., Katz, D. S., Berriman, G. B., Good, J., Laity, A. C., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., Prince, T. A., et al. (2010). Montage: An astronomical image mosaicking toolkit. 1:10036.

Juve, G., Chervenak, A. L., Deelman, E., Bharathi, S., Mehta, G., and Vahi, K. (2013). Characterizing and profiling scientific workflows. *Future Generation Comp. Syst.*, 29(3):682–692.

Liang, Q. A. and Zhao, J. L. (2008). Verification of unstructured workflows via propositional logic. In *Seventh IEEE/ACIS International Conference on Computer and Information Science (icis 2008)*, pages 247–252.

Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015). A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493.

Lopes, B., Benevides, M., and Haeusler, E. H. (2014). Extending Propositional Dynamic Logic for Petri Nets. *Elec. Notes in Theo. Comp. Science*, 305(11):67–83.

Marsan, M. A. (1990). Stochastic Petri Nets: An elementary introduction. In Rozenberg, G., editor, *Advances in Petri Nets 1989*, volume 424 of *LNCS*, pages 1–29. Springer.

Marsan, M. A. and Chiola, G. (1987). On Petri Nets with deterministic and exponentially distributed firing times. In *Advances in Petri Nets 1987*, pages 132–145. Springer.

Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., and Mattoso, M. (2011). An algebraic approach for data-centric scientific workflows. *PVLDB*, 4(12):1328–1339.

Salimifard, K. and Wright, M. (2001). Petri net-based modelling of workflow systems: An overview. *European Journal of Operational Research*, 134(3):664–676.

van der Aalst, W. M. P. (1998). The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66.

Zhao, L., Li, Q., Liu, X., and Du, N. (2009). A modeling method based on ccs for workflow. In *Proceedings of ICUIMC'09*, pages 376–384.