

Luís Vinícius Omar Baldissera

**Modelagem e simulação de relações entre
tarefas computacionais em ambiente de
nuvem**

São José do Rio Preto

2019

Luís Vinícius Omar Baldissera

Modelagem e simulação de relações entre tarefas computacionais em ambiente de nuvem

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do título de "Bacharel" em Ciência da Computação da UNESP

Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)

Instituto de Biociências, Letras e Ciências Exatas (IBILCE)

Bacharelado em Ciência da Computação

Prof. Dr. Aleardo Manacero Junior

São José do Rio Preto

2019

Luís Vinícius Omar Baldissera

Modelagem e simulação de relações entre tarefas computacionais em ambiente de nuvem

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para obtenção do título de "Bacharel" em Ciência da Computação da UNESP

Prof. Dr. Aleardo Manacero
Junior

Luís Vinícius Omar
Baldissera

Banca Avaliadora:

Prof. Dra. Renata Spolon Lobato

Prof. Dr. Rodolfo Meneguette

São José do Rio Preto

2019

Agradecimentos

Agradeço a Deus, que permitiu que eu chegasse até este momento com saúde física e mental¹.

Ao meu pai, Carlos Roberto Baldissera, e a minha mãe, Gislaine Shafica Omar Baldissera, pelo incansável apoio durante esses anos de graduação, e ao meu irmão, João Victor Omar Baldissera, pelo companheirismo.

Ao meu amor, Lara Libralão, pela força durante todo esse período e em especial na pandemia.

Aos meus colegas de graduação que estiveram comigo em todos os momentos, até naqueles com meia hora de antecedência². Em especial àqueles que tive a oportunidade de estar mais próximo: Beatriz Lima, Giovana Giardini, Igor Martinez e João Otávio.

Aos meus companheiros de laboratório, Lucas Cazarote (Tesouro), Fernanda Peronaglio, Fabio Amorim, Camila Tomashiro, Maicon Monttoso, e João Magri.

A todos os professores do departamento, em especial ao Prof. Dr. Aleardo Manacero, Profa. Dra. Renata Spolon e Profa. Dra. Carina Rondini.

À UNESP pela oportunidade dessa graduação e pela estrutura que me proporcionou esse aprendizado. E à FAPESP pelo fomento em meu projeto de iniciação científica que precedeu este trabalho de conclusão.

Ao leitor que está dedicando seu tempo para ler este trabalho³.

¹ Este trabalho foi realizado na pandemia. Saúde é uma dádiva!

² Tenho certeza que entenderão a referência

³ Caso o leitor já tenha sido citado, sinta-se lembrado duas vezes

*"Comece fazendo o que é necessário, depois o que é possível,
e de repente você estará fazendo o impossível"*

- São Francisco de Assis

Resumo

A computação em nuvem se apresenta cada vez mais presente no cotidiano dos desenvolvedores e dos usuários finais. Sua característica escalável, auto provisionada e seu atendimento sob demanda permitiram a aplicação de sistemas mais complexos cuja carga de trabalho pode ser caracterizada por um conjunto de tarefas interdependentes, em forma de *workflow*. Essa óptica incentivou a busca por metodologias cientes de dependências para otimizar a execução da carga no ambiente de nuvem. Tais métodos necessitam ser validados, o que pode ser feito com custo reduzido e maior flexibilidade utilizando simulação. No entanto os simuladores de nuvem atuais ou não oferece a modelagem de *workflow* ou exige do usuário conhecimento de linguagem específica para fazê-la. Neste contexto, este trabalho propõe uma abordagem gráfica para a modelagem de *workflows* e sua implementação no Simulador icônico de sistemas Paralelos e Distribuídos (iSPD), o qual já oferecia recurso para modelagem do *hardware* físico e virtualizado do ambiente de nuvem, mas até então não disponibilizava possibilidade de especificar dependências entre as tarefas da carga de trabalho. Os resultados obtidos mostram que a modelagem de *workflows* seguindo a abordagem icônica do iSPD produz simulações precisas, permitindo ao analista a experimentação sobre tarefas em nuvem com dependências entre si.

palavras-chave: Workflow, Simulação, Computação em Nuvem, Sistemas Distribuídos.

Abstract

Cloud computing is increasingly present in developers and end users' daily lives. Its scalable, self-provisioning and self-service features allowed more complex applications which workload can be characterized by a set of interdependent tasks, in the form of workflow. This perspective encouraged researches for dependency-aware methodologies to optimize the load execution in the cloud environment. Such methods need to be validated, which can be done with reduced cost and greater flexibility using simulation. However, current cloud simulators either do not offer workflow modeling or require specific language knowledge from the end user to do so. In this context, this work proposes a graphical approach to workflow modeling and its implementation in **iconic Simulator of Parallel and Distributed systems (iSPD)**, which already offered the cloud system's physical and virtual hardware modeling using graphical interface, but until then it did not provide the possibility to specify dependencies between the workload tasks. Results achieved show that modeling workflows through the iconic approach used in iSPD produces accurate simulations, allowing the analyst to experiment with dependent tasks in the cloud's model.

keywords: Workflow, Simulation, Cloud Computing, Distributed Systems.

Lista de ilustrações

Figura 2.1 – Pilha de serviços da computação em nuvem.	15
Figura 2.2 – Taxonomia do <i>design</i> de <i>workflow</i>	17
Figura 2.3 – Exemplo de modelagem com DAG.	17
Figura 2.4 – Exemplo de diagrama de roteamento.	18
Figura 2.5 – Exemplo de modelagem de <i>workflow</i> utilizando Rede de Petri.	19
Figura 2.6 – Modelagem de <i>workflow</i> utilizado UML.	19
Figura 2.7 – Modelagem utilizando LOTOS.	20
Figura 2.8 – Classificação dos sistemas de filas.	23
Figura 2.9 – Diagrama conceitual do iSPD. <i>RF</i> : Rede de Filas.	25
Figura 2.10–Funcionamento do motor de simulação do iSPD.	26
Figura 3.1 – Tarefa Atômica.	29
Figura 3.2 – Espera Temporizada.	30
Figura 3.3 – Ponto de Sincronização.	30
Figura 3.4 – Ponto de Ativação.	30
Figura 3.5 – Desvio Probabilístico.	30
Figura 3.6 – Ponto de Recursão.	31
Figura 3.7 – Diagrama de funcionamento da prefixação.	31
Figura 3.8 – Diagrama de funcionamento da precedência.	32
Figura 3.9 – Diagrama de funcionamento do tratamento de falha.	32
Figura 3.10–Expansão do <i>workflow</i> por 3 vias paralela.	33
Figura 3.11–Expansão do <i>workflow</i> por 3 iterações.	33
Figura 3.12–Expansão recursiva do <i>workflow</i> por 2 níveis.	33
Figura 3.13–Tela inicial do iSPD.	34
Figura 3.14–Janela de escolha da carga.	34
Figura 3.15–Janela de geração de carga.	35
Figura 3.16–Janela para edição de <i>workflow</i>	35
Figura 3.17–Diagrama UML do pacote que armazena o <i>workflow</i> na memória.	36
Figura 3.18–Diagrama UML da Cadeia de tarefas.	38
Figura 4.1 – Configuração do <i>workflow</i> Epigenomics.	41
Figura 4.2 – Configuração do <i>workflow</i> Sipht.	41
Figura 4.3 – Configuração do <i>workflow</i> Broadband.	42

Lista de abreviaturas e siglas

GSPD	Grupo de Sistemas P aralelos e D istribuídos
iSPD	<i>iconic Simulator of Parallel and Distributed systems</i>
NIST	<i>National Institute of Standarts and Technology</i>
SOA	<i>Service Oriented Architecture</i>
IaaS	<i>Infasyucture as a Service</i>
PaaS	<i>Platform as a Service</i>
SaaS	<i>Software as a Service</i>
M&S	Modelagem e Simulação
FIFO	<i>First In, First Out</i>
DAG	<i>Directed Acyclic Graph</i>
WfMC	<i>Workflow Management Coalition</i>
MFLOP	Milhões de Operações de Ponto Flutuante
MFLOPS	Milhões de Operações de Ponto Flutuante por Segundo
XML	<i>eXtensible Markup Language</i>
LOTOS	<i>Language Of Temporal Ordering Specification</i>
DCSim	<i>Data Centre Simulation</i>
Ns2	<i>Network simulator 2</i>
MPI	<i>Message Passsing Interface</i>
OMNeT	<i>Object-oriented Modular Network Simulator</i>

Sumário

1	INTRODUÇÃO	11
1.1	Justificativa e Motivação	11
1.2	Objetivos	12
1.3	Metodologia	12
1.4	Exequibilidade	13
1.5	Organização do texto	13
2	FUNDAMENTAÇÃO TEÓRICA	14
2.1	Computação em Nuvem	14
2.2	Workflows	16
2.2.1	DAG	16
2.2.2	Roteamento	17
2.2.3	Redes de Petri	18
2.2.4	Diagrama de Estados UML	19
2.2.5	LOTOS	19
2.3	Modelagem e Simulação	21
2.3.1	Simulação orientada a eventos discretos	21
2.4	Modelagem utilizando sistemas de filas	22
2.5	Simuladores de sistemas de computação em nuvem	23
2.5.1	CloudSim	23
2.5.2	iCanCloud	24
2.5.3	GreenCloud	24
2.5.4	DCSim	24
2.5.5	iSPD	25
2.5.6	Motor de simulação do iSPD	26
2.6	Trabalhos relacionados	26
2.7	Comparativo entre os simuladores	27
2.8	Considerações finais	28
3	METODOLOGIA	29
3.1	Abordagem gráfica proposta	29
3.2	Novas interfaces gráficas	33
3.3	Arquitetura de implementação	35
3.3.1	Modelo Icônico	35
3.3.2	Transformação da Cadeia de Clientes	36
3.3.3	Processamento pelo motor de simulação	37

3.4	Considerações finais	38
4	TESTES	40
5	CONCLUSÃO	43
5.1	Trabalhos Futuros	43
	REFERÊNCIAS	44
A	MICRODADOS DOS EXPERIMENTOS REALIZADOS	47

1 Introdução

Sistemas de computação em nuvem têm se tornado uma realidade cada vez mais presente, uma vez que permite o desenvolvimento de aplicações mais complexas e diversificadas. Isso ocorre pois o provisionamento automático de recursos que os provedores desses sistemas oferecem permite que os desenvolvedores mantenham seus esforços no desenvolvimento, enquanto contam com um serviço de infraestrutura de *hardware* confiável.

Com esse aumento de complexidade, o modelo de suas cargas computacionais não pode mais ser abstraído apenas para um conjunto de tarefas independentes. A ocorrência de trocas de mensagens entre tarefas em sistemas distribuídos faz com que o conjunto das tarefas que compõe a carga seja melhor representado em uma estrutura de *workflow*, que considera também as dependências de dados e de controle entre as tarefas (CALZAROSSA et al., 2016). A partir dessa visão são desenvolvidos métodos cientes das dependências do *workflow*, como escalonadores e agrupadores de tarefas (WENG et al., 2008; CHEN et al., 2013), que visam aumentar o desempenho da execução da carga computacional em sistemas de nuvem.

Para avaliar a melhoria de desempenho proposta por esses métodos, os mesmos necessitam ser validados. Isso pode ser feito com testes de desempenho em sistemas reais, o que acarreta em um certo desperdício de recursos do sistema de computação em nuvem. Pode também ser realizado com experimentação em um ambiente simulado que represente uma aplicação modelada e extraia as métricas pertinentes à avaliação de desempenho proposta.

Outro aspecto vantajoso para o uso de simulação é que a validação de diferentes configurações é mais simples e rápida se aplicada em um modelo do que em um ambiente real. A principal desvantagem da simulação é sua menor precisão em relação a medições no *hardware* real.

Porém, para que seja possível a realização dessa experimentação em ambiente simulado, é necessária a existência de um simulador que permita a modelagem da carga computacional em forma de *workflow*. É neste contexto de desenvolvimento de simuladores capazes de modelar e experimentar cargas em formato de *workflow* submetidas a ambientes de nuvem que este trabalho se insere.

1.1 Justificativa e Motivação

Dada a importância da simulação como forma de validar metodologias que são fundamentais para o bom funcionamento de sistemas de nuvem, como já explicado, é que se percebe a relevância do desenvolvimento de simuladores capazes de executar modelos com características de *workflow*.

Como será descrito no próximo capítulo (seção 2.5), o cenário de simuladores para ambientes de nuvem apresenta uma deficiência quanto a criação de modelos confiáveis de carga em forma de *workflow*. Os simuladores mais utilizados ou não apresentam a possibilidade de se modelar *workflows*, e os trabalhos relacionados que implementam a modelagem de *workflows* nesses simuladores exigem que o usuário crie o modelo em alguma linguagem específica para descrição do modelo de cargas.

A necessidade do conhecimento de uma linguagem traz outros custos em relação tanto ao aprendizado dessa linguagem quanto ao processo de modelar em si, os quais poderiam ser reduzidos se essa modelagem fosse facilitada. E uma forma de facilitar este processo é com a utilização de uma interface gráfica para modelar.

A partir dessas considerações se apresenta a motivação deste trabalho, que é *como modelar e simular cargas em formato de workflow que executam na nuvem de forma facilitada para o usuário*.

1.2 Objetivos

Este trabalho tem como objetivo principal propor uma abordagem gráfica para a modelagem de cargas na forma de *workflow*.

Para atingir e validar essa proposta, este trabalho tem como objetivo mais específico implementar a modelagem de *workflow* no simulador de sistemas paralelos e distribuídos iSPD (*iconic Simulator of Parallel and Distributed systems*), desenvolvido no GSPD (Grupo de Sistemas Paralelos e Distribuídos). Tal proposta busca atingir o seguinte escopo:

- Permitir a modelagem de forma amigável, sem a necessidade do conhecimento de linguagem específica pelo usuário final;
- Viabilizar a modelagem de cargas mais complexas em estruturas de *workflow*;
- Apresentar, por meio de seus algoritmos, a lógica necessária para a aplicação do modelo em simulação;
- Validar o modelo e a implementação proposta, comparando resultados simulados com resultados obtidos a partir de testes em sistemas reais.

1.3 Metodologia

Inicialmente é levantado o estado da arte dos simuladores de nuvem, e são avaliados seus requisitos de modelagem utilizando-se interface gráfica, e possibilidade de modelagem de *workflows*. Além disso, também são levantadas as formas de se representar tarefas em

estruturas de *workflow* que a literatura utiliza, tanto aquelas consideradas pelos métodos cientes de dependência que podem ser aplicados na nuvem quanto outras formas existentes de representação.

A partir dessa revisão, são identificadas as características relevantes para a modelagem de tarefas no simulador, envolvendo suas relações de dependências. Então é desenvolvida uma especificação de ícones para uma abordagem gráfica baseada nas representações da literatura, a qual é implementada no simulador iSPD.

Tendo a modelagem e sua execução disponível no iSPD, são realizados experimentos no simulador e seus resultados confrontados com testes em sistemas reais tendo finalidade de validar o funcionamento do projeto.

1.4 Exequibilidade

Este projeto é desenvolvido no laboratório do GSPD da UNESP, cuja estrutura permite a execução das etapas descritas na metodologia, tanto em sua parte teórica, devido ao amplo acesso a trabalhos disponíveis por meio da rede institucional e aos materiais da biblioteca, quanto em sua parte prática, uma vez que a estrutura do laboratório é suficiente para os experimentos no simulador. Além disso, este trabalho também contou com fomento de bolsa de iniciação científica da FAPESP, e foi apresentado na XI ERAD (Escola Regional de Alto Desempenho) (BALDISSERA et al., 2020). E por fim, o iSPD, que se trata do *software* no qual este trabalho é implementado, conta com outros membros trabalhando em conjunto, o que permite que seus detalhes de implementação sejam melhor discutidos.

1.5 Organização do texto

O restante deste texto está organizado de forma a que no capítulo 2 se apresente a fundamentação teórica na qual este trabalho se insere. Já no capítulo 3 é apresentado o processo da implementação dos mecanismos para modelagem e simulação de *workflows* no simulador iSPD. Em seguida, no capítulo 4, são apresentados alguns experimentos executados no simulador com a análise de seus resultados. Por fim, no capítulo 5 as conclusões são apresentadas e os trabalhos futuros sugeridos.

2 Fundamentação Teórica

Este trabalho se insere no contexto de simulação de tarefas computacionais com dependências entre si em um ambiente de nuvem. Visto isso, nas próximas seções são apresentados os conceitos envolvidos, sendo eles: Computação em nuvem, *Workflows*, modelagem e simulação, sistemas de filas, simuladores de computação em nuvem, evidenciando neste último alguns trabalhos relacionados em cada simulador.

2.1 Computação em Nuvem

A computação em nuvem é definida pelo NIST (*National Institute of Standards and Technology*) (MELL; GRANCE et al., 2011) como um modelo de computação que fornece ubiquidade e acesso sob demanda a recursos de rede, processamento, memória, armazenamento, entre outros. Para atingir essas máximas algumas características essenciais de um sistema de nuvem são capacidade do auto atendimento do consumidor, disponibilidade dos recursos de forma elástica, transparência de localidade desses recursos e um serviço que permita a mensuração da utilização dos recursos.

Uma forma de atingir essas propriedades é encapsular os recursos a serem provisionados em serviços ou domínios com funcionalidades bem definidas que se comunicam entre si, estratificando o provisionamento de rede, armazenamento, processamento e memória em entidades separadas que acessam recursos compartilhados. Essa abordagem foi utilizada por modelos predecessores da computação em nuvem, como o SOA (*Service Oriented Architecture*) e a computação em grade.

Nessa situação, a evolução das tecnologias de virtualização permitiu a construção de um ambiente computacional virtualizado sobre um ambiente físico distribuído, o que facilitou a implementação dos requisitos de elasticidade e escalabilidade do ambiente de nuvem. Por isso o modelo arquitetural da computação em nuvem pode ser considerado uma evolução natural dos *datacenters*, como notado por Buyya (BUYYA; BROBERG; GOSCINSKI, 2010). Nesse contexto, alguns exemplos de tecnologia de virtualização são KVM e Xen (BINU; KUMAR, 2011).

A partir da presença da virtualização, os sistemas de computação em nuvem podem ser divididos em três classes de acordo com o nível de abstração oferecido pelo provedor do sistema:

- **IaaS (Infraestrutura como Serviço)** fornece ao usuário gerenciamento direto dos recursos virtualizados de *hardware* e rede por meio do provisionamento de máquinas virtuais que podem ser acessadas remotamente. Exemplos de serviços de nuvem desse tipo são

AWS EC2([AMAZON, 2020](#)), Azure IaaS([MICROSOFT, 2020](#)) e Google Cloud Computing Engine([GOOGLE, 2020b](#)). Já alguns exemplos de *software* que implementam esse modelo e podem ser implantados são OpenStack([SEFRAOUI; AISSAOUI; ELEULDJ, 2012](#)) e OpenNebula([MILOJIČIĆ; LLORENTE; MONTERO, 2011](#)).

- **PaaS (Plataforma como Serviço)** fornece um ambiente de desenvolvimento e/ou implantação de *software* para o usuário, abstraindo os detalhes de infraestrutura. Um exemplo de PaaS é o Google AppEngine([GOOGLE, 2020a](#)).
- **SaaS (Software como Serviço)** são sistemas hospedados em ambiente de nuvem e acessados pelos usuários finais pela internet, abstraindo-se o desenvolvimento e a infraestrutura do sistema. Exemplos de SaaS são processadores de texto, planilha, apresentação, entre outras aplicações com alta disponibilidade *online*.

Tal categorização pode ser representada como uma pilha de classes de serviço, tendo como base a IaaS, e como topo o SaaS, como mostrado na figura 2.1.

Figura 2.1 – Pilha de serviços da computação em nuvem.

Classe de serviço	Acesso & Gerenciamento	Conteúdo de serviço
SaaS	Navegador web	Aplicações Redes sociais, pacotes <i>office</i> , processadores de vídeos
PaaS	Ambiente de desenvolvimento em nuvem	Plataforma Linguagens de programação, <i>frameworks</i> , editores, dados estruturados
IaaS	Gerenciador de infraestrutura virtual	Infraestrutura Servidores, armazenamento, <i>firewall</i> , balanceador de carga

Fonte: Adaptado de Buyya ([BUYYA; BROBERG; GOSCINSKI, 2010](#))

Já do ponto de vista de organização administrativa, os sistemas de nuvem são divididos em nuvens públicas, privadas e híbridas. Os sistemas públicos são aqueles em que o usuário o acessa pela *internet* em um modelo de negócio *pay as you go*, isto é, o custo do cliente é calculado dinamicamente com base na utilização dos recursos. Já os privados são normalmente implantados em empresas para uso interno. Por fim, os sistemas de nuvem híbridos são implantações que permitem ambas as estruturas.

Essa organização permite que o usuário final conte com uma infraestrutura auto provisionada para o desenvolvimento e implantação de aplicações mais complexas, as quais podem ser vistas em forma de *workflow*.

2.2 Workflows

Um *workflow*, quando associado ao modelo de carga computacional de uma aplicação, se refere à estrutura arquitetural da carga de trabalho, isto é, seu conjunto de tarefas e as dependências entre elas.

Tais dependências podem ser caracterizadas como dependências de controle ou de dados. Enquanto aquelas de controle se referem à sincronização e ordenação das tarefas, as dependências de dados acrescentam transferências de mensagem na execução do *workflow*. Uma visão mais detalhada sobre os tipos de dependências pode ser encontrada no trabalho de Kim (KIM, 2003).

A partir disso, a taxonomia de *workflow* pode ser vista de três pontos distintos:

- A estrutura, que se refere ao comportamento temporal de suas dependências quando o *workflow* é executado, podendo ser caracterizada como DAG, quando não permitir ciclos; e como “não DAG”, quando há a possibilidade de ciclos.
- O modelo, ou especificação, que toca a respeito do nível de abstração na definição do *workflow*, podendo ser abstrato, quando a definição não inclui o local em que a carga será executada, ou concreto, quando o inclui.
- A composição, que denota como o processo de especificação é realizado, podendo ser automático se o *workflow* é gerado a partir de requisitos de alto nível, ou dirigido pelo usuário se é modelado a partir da especificação de um grafo ou utilizando uma linguagem de programação.

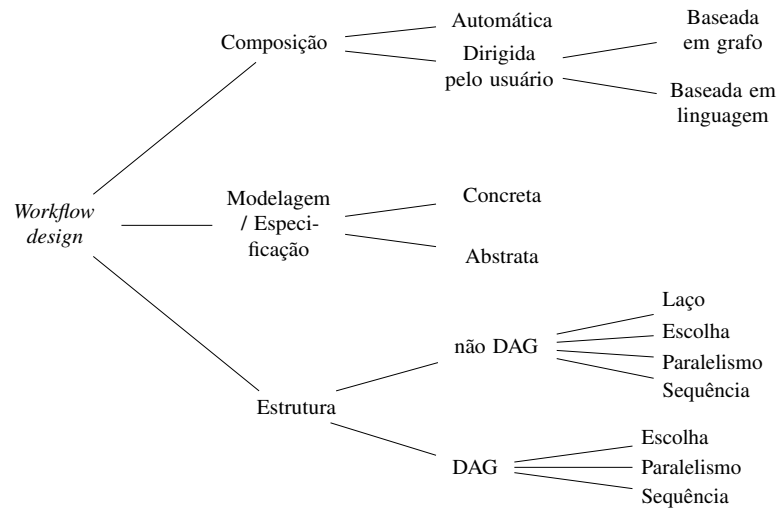
Essa taxonomia é exibida na figura 2.2.

Aprofundando-se nos modelos que os trabalhos da literatura utilizam para a especificação de *workflows*, as principais representações são descritas a seguir.

2.2.1 DAG

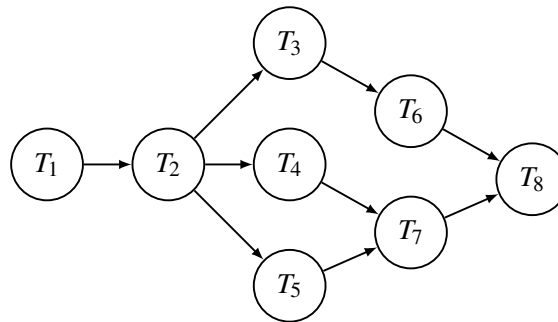
O Grafo Acíclico Direcionado (DAG) é a principal representação do *workflow* na literatura. Nesse modelo, os vértices denotam as tarefas e as arestas representam as dependências entre elas, possivelmente com transferências de mensagem. Na figura 2.3 é apresentado um exemplo de DAG.

Também é possível realizar pequenas alterações na descrição do DAG para atingir maior especificidade dos *workflows*. Por exemplo, no trabalho de Marzakis (MARZAKIS; NIKOLAOU,

Figura 2.2 – Taxonomia do *design* de *workflow*.

Fonte: Adaptado de Yu (YU; BUYYA, 2005)

Figura 2.3 – Exemplo de modelagem com DAG.



Fonte: Retirado de Calzarossa (CALZAROSSA et al., 2016)

1995) o conjunto de vértices modelado é particionado entre tarefas primárias e secundárias, e no trabalho de Chen (CHEN et al., 2013) é dividido em tarefas e *overheads* de sincronização. Já no trabalho de Elwasif (ELWASIF et al., 2011) o conjunto de arestas é particionado com base no tipo de dependência que representa.

Portanto o DAG apresenta como vantagens a visualização facilitada das relações entre as tarefas e a flexibilidade da sua especificação. No entanto, uma desvantagem intrínseca dessa representação é a impossibilidade de se modelar ciclos.

2.2.2 Roteamento

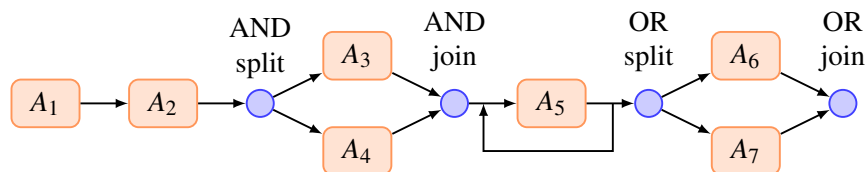
O diagrama de roteamento se baseia em quatro conceitos básicos definidos pela Coligação de Gerenciamento de *Workflow* (WfMC) (AALST; HEE; HEE, 2004):

- **Roteamento sequencial** representa uma atividade que deve ocorrer em sequência de outra.

- **Roteamento paralelo** diz respeito à modelagem de duas ou mais atividades que ocorrem simultaneamente. Nesse caso, as atividades paralelas devem ser iniciadas por um *AND-split* e finalizadas por um *AND-join*.
- **Roteamento seletivo ou condicional** modela um conjunto de atividades em que a ocorrência de uma impeça as demais. Esse tipo de roteamento é iniciado com um *OR-split* e finalizado com um *OR-join*.
- **Roteamento iterativo** permite a modelagem da repetição iterativa da execução de atividades.

Tendo esses conceitos em vista, um diagrama de roteamento representa as tarefas da carga de trabalho como atividades, e suas relações como roteamento. Na figura 2.4 é exemplificado uma carga na qual a atividade A_1 deve ocorrer antes de A_2 , então A_3 e A_4 podem ocorrer simultaneamente, e quando ambas forem concluídas A_5 deve ocorrer repetidamente um determinado número de vezes e então deve ocorrer A_6 ou A_7 , mas não ambas.

Figura 2.4 – Exemplo de diagrama de roteamento.



Fonte: Adaptado de Tick (TICK, 2006)

Essa metodologia apresenta como vantagem a modelagem de ciclos e desvios condicionais, e a detecção de alguns possíveis *deadlocks* no modelo, os quais ocorrem quando as ramificações criadas por um *OR-split* são juntadas por um *AND-join* (LIU; KUMAR, 2005).

2.2.3 Redes de Petri

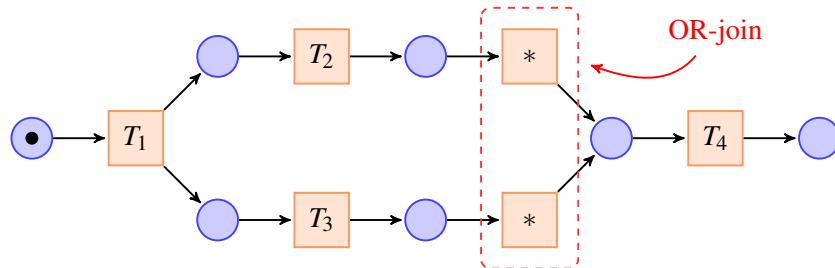
A Rede de Petri, definida em 1962 por Carl Adam Petri no contexto de análise de processos é uma modelagem formal que pode ser utilizada para a representação de *workflow*, sendo capaz de representar os tipos de roteamento definidos anteriormente.

Ela consiste de locais que são representados por círculos, e transições que são representadas por retângulos. Os locais só devem ser ligados a transições, e transições a locais, por meio de arcos direcionados. Cada local pode conter *tokens*, e quando todos os locais que se ligam a uma determinada transição têm pelo menos um *token*, então essa transição é habilitada, consumindo um *token* dos locais anteriores e gerando um novo *token* em cada local posterior.

Dessa forma, as tarefas de um *workflow* podem ser modeladas como transições, enquanto as dependências, pontos de controle e sincronização por meio dos locais e *tokens*. Na figura 2.5 é

mostrado um exemplo de modelagem de um *workflow* de quatro tarefas. Nota-se que a partir de T_1 há um *AND-split* e são utilizadas duas transições marcadas com asterisco para a modelagem de um *OR-join*.

Figura 2.5 – Exemplo de modelagem de *workflow* utilizando Rede de Petri.



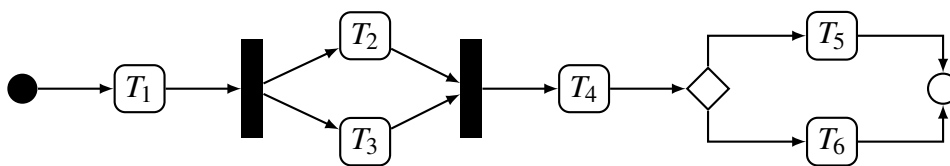
Fonte: Adaptado de Tick (TICK, 2006)

2.2.4 Diagrama de Estados UML

O diagrama de estados da Linguagem de Marcação Unificada (UML) também pode ser utilizado para a representação de *workflow*. Neste caso, os estados podem ser utilizados para representar as tarefas; as barras de *fork* e *join* para a modelagem de *AND-split* e *AND-join*, respectivamente, e os pseudo-estados de escolha para o comportamento de *OR-split* e *OR-join*.

Na figura 2.6 é mostrado um exemplo dessa modelagem. A tarefa T_1 deve ocorrer, então T_2 e T_3 podem ocorrer simultaneamente, quando ambas forem concluídas então T_4 deve ocorrer, e por fim T_5 ou T_6 deve ocorrer, mas não ambas.

Figura 2.6 – Modelagem de *workflow* utilizando UML.



Fonte: Adaptado de Tick (TICK, 2006)

2.2.5 LOTOS

No formalismo LOTOS (Linguagem de Especificação de Ordenação Temporal) um sistema pode ser modelado como um processo, possivelmente composto de outros sub-processos, no qual cada processo ou sub-processo realiza um conjunto de ações, ou eventos. Esses eventos provocam sincronização e possivelmente troca de dados. (BOLOGNESI; BRINKSMA, 1987)

A notação é dividida em dois grupos: LOTOS básica e LOTOS completa. Enquanto a básica trata a respeito da sincronização dos processos, a completa acrescenta a comunicação entre eles. Da notação LOTOS básica, destaca-se:

$a; B$: Diz-se que a ação a é prefixa do processo B , ou seja, o processo B deve ocorrer após a ação a . No entanto, isso não implica que o processo que contém a ação a seja concluído com êxito.

$B||C$: Denota uma relação de paralelismo entre os processos B e C , em que ambos podem executar no mesmo momento e o encerramento de suas execuções deve ser sincronizado.

$B[]C$: Há uma relação de escolha entre os processos B e C , de forma que ou B ou C deve ocorrer, mas não ambos.

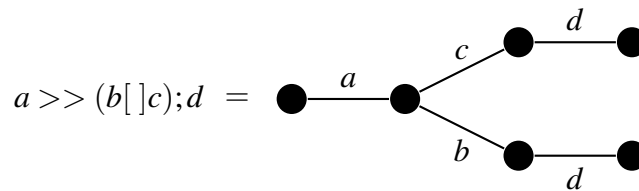
$B >> C$: Demonstra uma relação de precedência, em que o processo C deve ocorrer somente após B ter sido concluído com êxito.

$B[> C$: Diz respeito a uma relação de tratamento de falha, na qual a falha do processo B habilita a execução do processo C .

A partir da formalização com a sintaxe apresentada, pode ser construída uma árvore de ações que descreve consistentemente o comportamento do sistema, e consequentemente, modelar *workflows* computacionais.

Na figura 2.7 é exemplificado um sistema em que a ação a deve ocorrer e ser concluída, então ou b ou c deve ocorrer, mas não ambas, logo antes da ocorrência de d . Essa construção permite duas ordenações de eventos: $a \rightarrow b \rightarrow d$, ou $a \rightarrow c \rightarrow d$.

Figura 2.7 – Modelagem utilizando LOTOS.



Fonte: Elaborado pelo autor.

Tendo as relações entre as tarefas do *workflow* definidas a partir desses modelos, um ambiente de computação distribuída pode otimizar a execução do *workflow* com abordagens cientes de dependência, as quais são utilizadas por sistemas de gerenciamento de *workflows*. Exemplos desses sistemas são Pegasus (DEELMAN et al., 2015) e Apache-Airflow (AIRFLOW, 2020).

Tais métodos cientes de dependência necessitam ser validados, o que pode ser realizado por simulação que traz vantagens em relação ao baixo custo de manutenção e flexibilidade de configuração, como é exposto a seguir.

2.3 Modelagem e Simulação

A metodologia de modelagem e simulação (M&S) se trata da abstração de características relevantes de um sistema real para um modelo simulado que pode ser livremente experimentado para a obtenção de métricas pertinentes a sua investigação, e que reflitam os resultados esperados no sistema real. Desse modo, a principal vantagem da M&S é a flexibilidade de se experimentar um ambiente hipotético e alterá-lo sem a imposição de sua testagem real.

A partir dessa perspectiva, os termos “modelagem” e “simulação” podem ser melhor discernidos(LOUIS; GILBERT, 2007):

- A **modelagem** diz respeito à especificação do comportamento do sistema que deve ser investigado, e por conseguinte o processo de modelagem se trata do desenvolvimento dessa especificação.
- Já **simulação** é o processo de experimentação do modelo previamente especificado e obtenção de resultados desses experimentos.

Nesse contexto, a utilização de M&S traz vantagens para a observação de um sistema em relação à testagem em ambiente real, a saber:

- **Custo.** A utilização de M&S, em geral, acarreta menor custo associado à configuração e manutenção do sistema;
- **Risco.** A tomada de decisões equivocadas pelo analista do sistema em um ambiente simulado não acarreta impacto direto no sistema real;
- **Flexibilidade.** A experimentação por intermédio da modelagem aumenta a flexibilidade da especificação do sistema, haja vista que uma alteração no sistema seja equivalente a uma alteração na especificação do modelo.

A partir deste âmbito, existem abordagens bem definidas para auxiliar desenvolvedores na implementação de M&S. A seguir será tratado de simulação orientada a eventos discretos que é adequada à natureza dos sistemas de computação em nuvem, no entanto existem outras abordagens para simulação como baseada em agentes (DAVIDSSON, 2000) e simulação de Monte Carlo (MOONEY, 1997).

2.3.1 Simulação orientada a eventos discretos

De acordo com Lemmis (LEEMIS; PARK, 2006), a simulação orientada a eventos discretos é aplicável quando a natureza do sistema a ser simulado é:

- **Não determinístico.** Várias execuções da simulação de um mesmo modelo não implicam em resultados iguais, isso se deve ao fato de pelo menos uma variável de estado do sistema simulado ser aleatória;
- **Dinâmico.** A evolução dos estados do sistema durante o tempo de simulação deve ser considerada;
- **Discreto.** Mudanças significativas do sistema são associadas com instantes específicos no tempo, não sendo necessário considerar mudanças de estados entre esses instantes.

Nesta abordagem o modelo recebe um conjunto de entrada de eventos discretos iniciais e sua simulação produz um conjunto de saída de eventos que descreve o comportamento do sistema sob aqueles eventos de entrada. Esse conjunto de saída pode então ser analisado para se extrair as métricas de interesse da simulação.

2.4 Modelagem utilizando sistemas de filas

A modelagem por meio de filas é aplicável quando o sistema pode ser visto a partir de um conjunto de recursos, denominados centros de serviço, que são utilizados por um conjunto de usuários, ou clientes. Dessa forma, a indisponibilidade de algum recurso por um período resulta em filas ou linhas de esperas ([SOARES, 1992](#)).

Isso ocorre na modelagem de clientes de um supermercado utilizando os caixas para pagamento; passageiros em pontos de ônibus; turistas em visitas, etc. E particularmente, também ocorre na modelagem de sistemas de computação, como pacotes que devem ser atendidos por enlaces de comunicação ou tarefas que devem ser executadas em processadores e máquinas distribuídas.

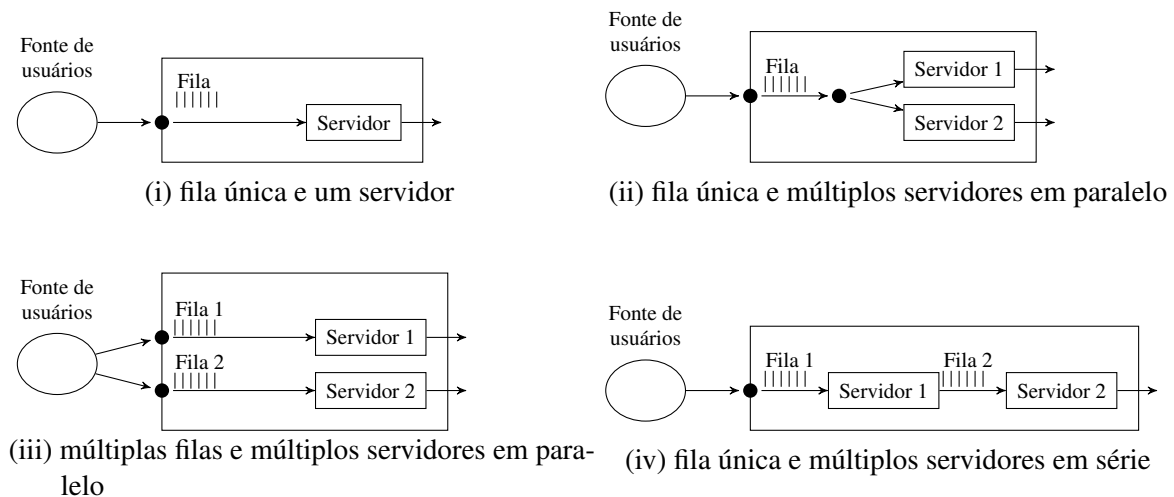
Em sistemas de filas, o atendimento de um cliente por um centro de serviço pode encaminhar o mesmo para outro centro de serviço, e possivelmente outra fila. Por outro lado, vários centros de serviço podem revezar entre si para o atendimento de uma mesma fila. Vale notar que cada fila pode ter sua própria disciplina definida, não sendo necessariamente uma fila do tipo FIFO (*First In, First Out*), mas também sendo possível, por exemplo, filas de prioridade.

Estes comportamentos acarretam a classificação do sistema de filas em quatro tipos: ([ARENALES; ARMENTANO et al., 2006](#), Cap. 7)

- (i) Fila única e um servidor
- (ii) Fila única e múltiplos servidores em paralelo
- (iii) Múltiplas filas e múltiplos servidores
- (iv) Fila única e múltiplos servidores em série

Além disso, a chegada desses clientes ao sistema também deve ser modelada por meio de uma fonte geradora de clientes que obedeça a alguma regra, como uma distribuição de probabilidade para indicar o momento da chegada dos clientes. Tal classificação está representada na figura 2.8.

Figura 2.8 – Classificação dos sistemas de filas.



Fonte: Adaptado de Arenales ([ARENALES](#); [ARMENTANO et al., 2006](#))

Por fim, uma rede de filas é definida por vários sistemas de filas interligados de forma que a saída de clientes de um sistema se torne a chegada em outro(s).

2.5 Simuladores de sistemas de computação em nuvem

Tendo em vista a utilização de simulação, a literatura traz simuladores de sistemas de nuvem que permitem a experimentação de diferentes cargas de trabalho em um ambiente simulado, possibilitando a obtenção de resultados interessantes sobre o teste de desempenho, como tempo de execução e ocupação de recursos, sem a necessidade de se fazê-lo no *hardware* real.

Em geral, esses simuladores se baseiam nos conceitos de simulação orientada a eventos discretos e/ou sistemas de filas apresentados anteriormente. Nesse contexto, alguns dos principais simuladores da área são apresentados a seguir, com enfoque na possibilidade de modelagem de *workflows* e suporte à interface gráfica.

2.5.1 CloudSim

O *CloudSim* é um *framework* para modelagem e simulação de sistemas de nuvem desenvolvido em Java ([CALHEIROS et al., 2011](#)). É baseado em simulação orientada a eventos discretos e sua modelagem ocorre por intermédio da codificação do ambiente simulado, o que

inclui a especificação da infraestrutura de nuvem, máquinas virtuais, e carga de trabalho. No entanto a carga de trabalho modelada não leva em consideração as dependências entre as tarefas, sendo vistas como tarefas independentes.

Existe também CloudAnalyst ([WICKREMASINGHE; CALHEIROS; BUYYA, 2010](#)) que é uma extensão para o CloudSim que permite a configuração de parâmetros de alto nível graficamente, mas que também não oferece a modelagem de *workflows*.

2.5.2 iCanCloud

O iCanCloud ([NÚÑEZ et al., 2012](#)) é uma plataforma de simulação em nuvem especialmente voltada, mas não restrita, à simulação de instâncias virtuais oferecidas pela AWS EC2.

É desenvolvido em C++ e oferece uma interface gráfica nativa implementada em Java. Seu processo de simulação se baseia no simulador de eventos discretos OMNeT (*Object-oriented Modular Network Simulator*) ([VARGA, 2001](#)) e na biblioteca para troca de mensagens MPI (*Message Passing Interface*) ([WALKER; DONGARRA, 1996](#)).

Do ponto de vista de carga de trabalho modelada, existem algumas aplicações pré-definidas no iCanCloud, e é oferecido uma forma de estender o próprio código para criar aplicações específicas do usuário. No entanto, os *jobs* gerados dessas aplicações são vistos como entidades independentes entre si, não sendo oferecida a modelagem direta de *workflows*.

2.5.3 GreenCloud

O GreenCloud ([KLIAZOVICH; BOUVRY; KHAN, 2012](#)) é um simulador de sistemas de nuvem com objetivo de avaliar métricas do consumo de energia em datacenters. É desenvolvido na linguagem C++ sobre a plataforma Ns2 (Network Simulator) ([NS, 2020](#)).

Sua modelagem de carga de trabalho utiliza parâmetros de alto nível como tamanho computacional da carga, tamanho de comunicação interna e externa, e o tipo do *workload*, podendo ser de alta intensidade computacional, de alta intensidade de dados, ou balanceado. Esse conjunto de carga é tratado de forma independente no simulador, não permitindo a modelagem da caracterização de um *workflow* específico.

2.5.4 DCSim

O DCSim ([TIGHE et al., 2012](#)) é um simulador com o objetivo de testar algoritmos para o gerenciamento de máquinas virtuais em nuvens IaaS.

É desenvolvido como um *framework* utilizando Java. Do ponto de vista de carga computacional, o simulador oferece nativamente a modelagem de cargas transacionais, e também

é possível se criar modelos de *workflow* estendendo-se classes do *framework* utilizando o próprio código. No entanto essa modelagem de *workflow* não é fornecida de forma nativa.

2.5.5 iSPD

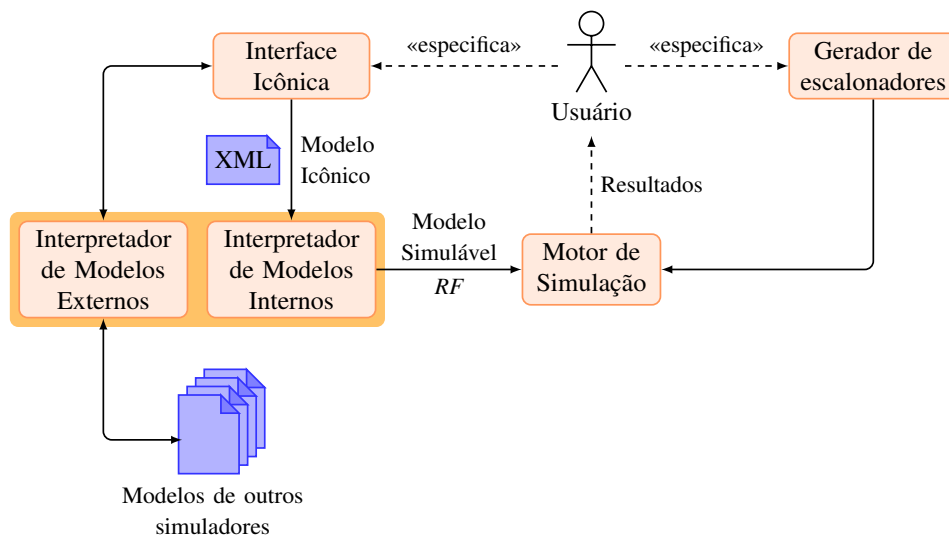
O simulador iSPD (*iconic Simulator of Parallel and Distributed Systems*) se baseia na simulação orientada a eventos discretos, e na modelagem com sistemas de filas, discutidos nas seções 2.3 e 2.4, respectivamente, para experimentar sistemas paralelos e distribuídos com ênfase em sistemas de computação em nuvem.

Foi primeiramente proposto em 2012 (MANACERO et al., 2012) com objetivo de permitir que o usuário possa modelar e simular sistemas de computação sem que necessite conhecimento de alguma linguagem específica. Para esse fim, conta com alguns módulos essenciais e suas interações com o usuário.

Inicialmente, o usuário modela o *hardware* e a carga de trabalho na Interface Icônica nativa. Esse modelo icônico especificado pelo usuário é persistido na memória em um arquivo XML (*extensible Markup Language*), o qual é transformado pelo Interpretador de Modelos Internos em um modelo simulado, isto é, uma sistema de rede de filas, como discutido na seção 2.4, que por sua vez é processado pelo Motor de Simulação, o qual retorna ao usuário os resultados pertinentes àqueles experimentos.

Ademais, o iSPD também oferece a possibilidade de importar e exportar arquivos para outros simuladores por meio do Interpretador de Modelos Externos, e conta com a criação de escalonadores especificados pelo usuário pelo Gerador de Escalonares. A interação entre esses módulos descritos e o usuário está exposto na figura 2.9.

Figura 2.9 – Diagrama conceitual do iSPD. RF: Rede de Filas.



Fonte: Adaptado de Silva et al (SILVA et al., 2015)

2.5.6 Motor de simulação do iSPD

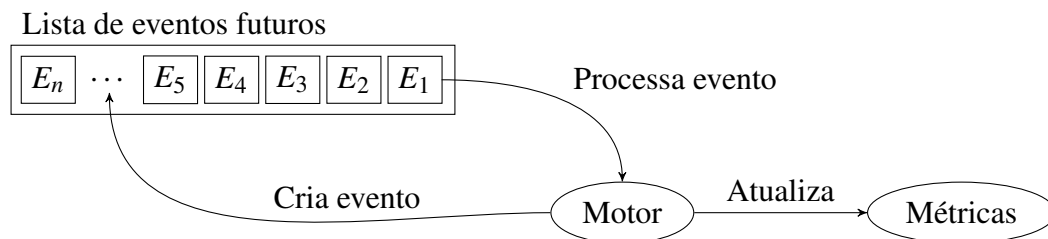
Um componente particularmente interessante de ser analisado é o motor de simulação do iSPD, que como citado anteriormente, é orientado a eventos discretos. Nesta seção será exposto com mais detalhes seu funcionamento.

Um evento discreto no iSPD é uma estrutura que contém as seguintes informações: o instante de ocorrência do evento, o tipo do evento, a tarefa ou mensagem que está sujeita ao evento e o componente de hardware em que o evento ocorre. Alguns exemplos são expostos a seguir:

- A tarefa t_5 (sujeito) começa a ser executada (tipo) na máquina virtual MV_2 (componente de hardware) no instante 1 minuto e 23 segundos
- A mensagem m_2 (sujeito) chega (tipo) no enlace e_3 (componente de hardware) no instante 3 minutos e 45 segundos

Estes eventos são colocados em uma lista de eventos futuros, a qual sempre se mantém ordenada com base no tempo de ocorrência de cada evento. Visto isso, o motor de simulação processa, em cada iteração, o próximo elemento da lista. Neste processamento o motor pode criar um novo evento futuro e/ou atualizar as métricas. Por fim, a simulação pode ser dada como concluída quando não há mais eventos a serem processados, ou seja, a lista de eventos futuros se encontra vazia. Este fluxo é exposto na figura 2.10.

Figura 2.10 – Funcionamento do motor de simulação do iSPD.



Fonte: Elaborado pelo autor.

2.6 Trabalhos relacionados

A partir da dificuldade ou impossibilidade da modelagem de *workflows* nos simuladores apresentados, a literatura traz trabalhos relacionados ao presente, que implementam essa modelagem. No entanto, tais trabalhos exigem do usuário o conhecimento de linguagem específica, enquanto o presente trabalho busca facilitar essa modelagem de forma gráfica.

O *WorkflowSim* (CHEN; DEELMAN, 2012), por exemplo, amplia o *CloudSim* para permitir a modelagem de *workflows*. Essa modelagem é feita a partir da codificação de um arquivo XML pelo usuário que especifica um DAG para a execução das tarefas. Não fornecendo, no entanto, a possibilidade de modelagem através de interface gráfica.

Existem também outras modelagens gráficas de dependências de tarefas como já apresentadas na seção 2.2, não necessariamente implementadas em simuladores, como o DAG, as redes de petri e os diagramas UML.

O DAG é o modelo mais utilizado na literatura e apresenta como vantagem a simplicidade de visualização. Por outro lado ele pode ter uma quantidade de ícones consideravelmente crescente conforme a quantidade de tarefas modeladas e a repetição de padrões de dependência das mesmas crescem. Este trabalho flexibiliza a modelagem do DAG inserindo ícones simplificados para abstrair tais padrões.

As redes de Petri e a notação LOTOS também permitem algumas modelagens não previstas no DAG, como *OR-joins* e *OR-split*, laços, *AND-join* e *AND-split* e independência de ordem. No entanto, ainda que mais flexíveis, exigem mais conhecimento para visualização por se tratarem de modelagens formais. Este trabalho busca simplificar essa visualização acrescentando ícones específicos no modelo que representam tais relações.

Por fim, o diagrama de fluxo UML permite também a modelagem do desvio condicional do *workflow*. A proposta deste trabalho prevê a possibilidade de desvios de fluxo baseado em uma probabilidade de ocorrência do fluxo para mimicar o desvio condicional em um DAG e manter sua aciclicidade.

2.7 Comparativo entre os simuladores

Na tabela 2.1 é mostrado um comparativo das características dos simuladores apresentados.

Tabela 2.1 – Comparação entre os simuladores avaliados considerando-se aspectos de facilidade na modelagem do ambiente de nuvem.

Simulador	Linguagem de Programação	Modelagem de <i>workflow</i>	Suporte a GUI
CloudSim	Java	Código	Através de Extensão
iCanCloud	C++/Java	-	Nativo
GreenCloud	C++	-	Nativo
DCSim	Java	Código	-
iSPD	Java	GUI*	Nativo

*Objetivo deste trabalho

Fonte: Elaborado pelo autor.

Como pode ser observado, o CloudSim oferece a modelagem de *workflow* através do código, porém ainda que o mesmo tenha suporte à interface gráfica através de extensão, não é possível realizar a modelagem de *workflows* utilizando esta interface.

Já o iCanCloud e o GreenCloud suportam interface gráfica de forma nativa, no entanto não oferecem a possibilidade de se modelar *workflows*. Em contraste, o DCSim fornece a possibilidade de se modelar *workflows* através do código, mas não dá suporte à interface gráfica.

Por fim, o iSPD dá suporte à interface gráfica, e até então não fornecia a modelagem de *workflows* através dessa interface, sendo este o objetivo deste trabalho e destacando a sua relevância em relação aos simuladores da literatura.

2.8 Considerações finais

Neste capítulo se apresentou a base conceitual para este projeto, o que inclui a definição de um sistema de nuvem e de *workflows* computacionais que são executados nesses sistemas; as diferentes formas de se modelar esses *workflows* e a importância da modelagem para se propor as abordagens cientes de dependência. Então foram apresentados os conceitos sobre modelagem e simulação, com enfoque na modelagem utilizando sistemas de filas e na simulação orientada a eventos discretos. Por fim, foram discutidos alguns dos principais simuladores de nuvem da literatura, com maior foco no iSPD que é o simulador objetivo deste trabalho.

3 Metodologia

Neste capítulo são abordadas as atividades desenvolvidas na realização do trabalho. Inicialmente foi feita uma revisão da literatura no método sistemático (KEELE et al., 2007) para entender como os *workflows* são modelados e caracterizados, o que é tratado na seção 2.2. À vista disso, foram implementadas novas funcionalidades no simulador iSPD para permitir a modelagem e simulação de *workflows*.

Na seção 3.1 é discorrida a abordagem gráfica proposta para a modelagem de *workflows*. Então na seção 3.2 as novas interfaces gráficas são apresentadas. Por fim, na seção 3.3 é detalhada a arquitetura e implementação da abordagem no iSPD.

3.1 Abordagem gráfica proposta

Para a representação do *workflow* na interface gráfica do iSPD foi definida uma abordagem baseada na modelagem com DAG. Nela os nós representam tipos de carga computacional e as arestas tipos de dependência. Além disso, cada um desses elementos tem uma *tag* associada para que seja possível armazená-lo em formato XML. A seguir é listada a definição, representação gráfica e *tag* XML de cada ícone.

- **Tarefa atômica:** é definida pela sua carga computacional, em MFLOP (Milhões de Operações de Ponto Flutuante), e opcionalmente pode ser inserida dentro um grupo de exclusão mútua de execução. Dessa forma, se existem várias tarefas em um mesmo grupo de execução, então podem ocorrer em qualquer ordem, mas não ao mesmo tempo, modelando a relação de independência de ordem. Sua representação é encontrada na figura 3.1.

Figura 3.1 – Tarefa Atômica.



Fonte: Elaborado pelo autor.

XML <task id,x,y,computing,lock>, onde *id* é o identificador do ícone, (*x,y*) é a posição do ícone na tela, *computing* é a carga computacional, e *lock* é o identificador do *lock*.

- **Espera temporizada:** representa atrasos, sendo determinada pelo seu tempo, em segundos. O ícone é mostrado na figura 3.2.

Figura 3.2 – Espera Temporizada.



Fonte: Elaborado pelo autor.

XML `<timer id, x, y, time>`, onde *id* é o identificador do ícone, (x, y) é a posição do ícone, *time* é o tempo de espera.

- **Ponto de Sincronização:** permite a modelagem de sincronização de execução do *workflow*, mais especificamente, *AND-joins* e *AND-splits*. Sua imagem é exibida na figura 3.3.

Figura 3.3 – Ponto de Sincronização.



Fonte: Elaborado pelo autor.

XML `<sync id, x, y>`, onde *id* é o identificador do ícone e (x, y) sua a posição.

- **Ponto de Ativação:** permite a modelagem de *OR-join* e *OR-split*. Sua representação é exposta na figura 3.4.

Figura 3.4 – Ponto de Ativação.



Fonte: Elaborado pelo autor.

XML `<act id, x, y>`, onde *id* é o identificador do ícone e (x, y) sua a posição.

- **Desvio Probabilístico:** permite a modelagem de desvios de fluxo no *workflow*. É configurado com as probabilidades de desvio para cada fluxo. Seu símbolo é apresentado na figura 3.5.

Figura 3.5 – Desvio Probabilístico.



Fonte: Elaborado pelo autor.

XML `<switch id, x, y>`, onde *id* é o identificador do ícone e (x, y) sua a posição. Cada *tag* é ainda composta internamente por um conjunto de *tags* `<flow edge, prob>` onde

edge é o identificador de uma aresta conectada ao *switch* e *prob* é a probabilidade do fluxo seguir o caminho desta aresta.

- **Ponto de Recursão:** permite a modelagem de chamadas recursivas em uma expansão, abordada mais adiante. Sua representação é indicada na figura 3.6.

Figura 3.6 – Ponto de Recursão.



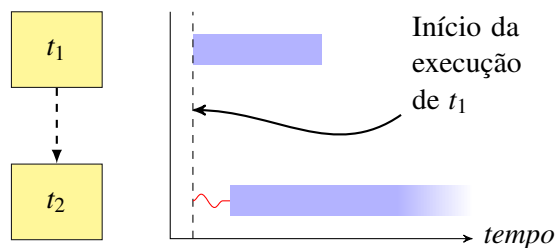
Fonte: Elaborado pelo autor.

XML `<recur id, x, y>`, onde *id* é o identificador do ícone e (*x, y*) sua a posição.

Até o momento os nós do DAG foram denotados. A seguir as arestas serão exibidas, sendo categorizadas segundo o tipo de relação que permite modelar. Todas elas podem ser configuradas com um tamanho de mensagem que é transferida entre os nós que conectam.

- **Prefixação:** permite a modelagem de uma dependência começo-começo, isto é, o sucessor pode ser iniciado em algum momento após o predecessor já ter sido iniciado. É representada por uma seta tracejada, como mostrado na figura 3.7.

Figura 3.7 – Diagrama de funcionamento da prefixação.

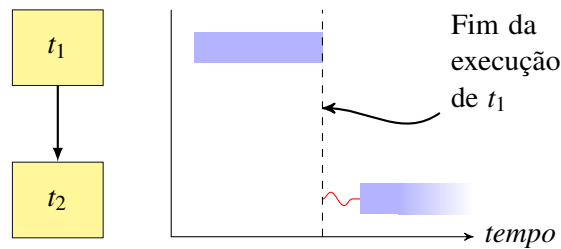


Fonte: Elaborado pelo autor.

XML `<prefix id, src, dest, message >`, onde *id* é o identificador do ícone e (*src, dest*) são os identificadores dos nós que a aresta conecta, e *message* é o tamanho da mensagem que é passada entre os nós que a aresta conecta.

- **Precedência:** permite a modelagem de uma dependência fim-começo, ou seja, o sucessor poderá iniciar a execução em algum momento após o predecessor finalizar a sua. É desenhada como seta contínua conforme exibido na figura 3.8

Figura 3.8 – Diagrama de funcionamento da precedência.

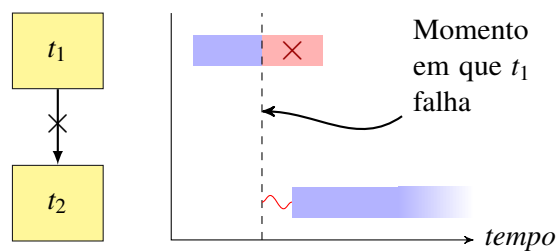


Fonte: Elaborado pelo autor.

XML `<precedence id,src,dest,message >`, onde *id* é o identificador, (*src,dest*) os identificadores dos nós que a aresta conecta, e *message* é o tamanho da mensagem.

- **Tratamento de falha:** permite a modelagem do tratamento de uma falha num ponto de execução. Sua simbologia é representada na figura 3.9.

Figura 3.9 – Diagrama de funcionamento do tratamento de falha.

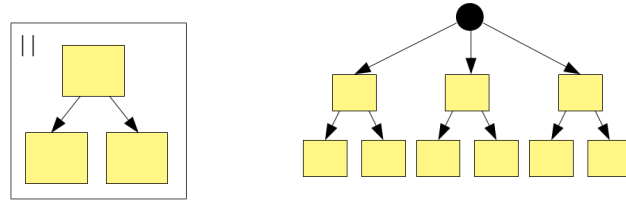


Fonte: Elaborado pelo autor.

XML `<fail id,src,dest,message >`, onde *id* é o identificador, (*src,dest*) os identificadores dos nós que a aresta conecta, e *message* é o tamanho da mensagem.

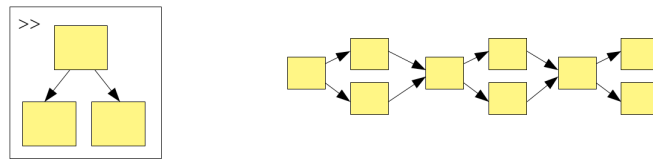
Os ícones apresentados até o momento permitem a modelagem de *workflows* de forma gráfica, no entanto algumas aplicações que podem ser compostas de dezenas ou milhares de tarefas exigiria que o usuário inserisse uma grande quantidade de ícones no *workflow*. Então pensando na usabilidade para esses casos são definidos os nós de expansão que possibilita a modelagem de cargas mais complexas a partir da especificação de um DAG interno, abstraindo alguns padrões da estruturação do *workflow*.

- A expansão **paralela** possibilita a modelagem de seu *workflow* interno como n blocos paralelos e pontos de sincronização são adicionados em seu começo e fim. Sua interpretação é denotada na figura 3.10.

Figura 3.10 – Expansão do *workflow* por 3 vias paralela.

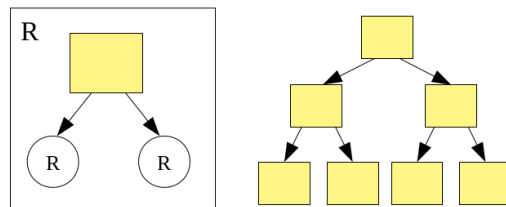
Fonte: Elaborado pelo autor.

- A expansão **iterativa** permite representar a repetição do *workflow* interno por n iterações, e são adicionados pontos de sincronização entre suas iterações. Seu sentido é exemplificado na figura 3.11.

Figura 3.11 – Expansão do *workflow* por 3 iterações.

Fonte: Elaborado pelo autor.

- A expansão **recursiva** propicia a modelagem de chamadas recursivas do próprio *workflow* de forma que a árvore de recursão não ultrapasse n níveis. Os pontos onde as chamadas recursivas devem ocorrer são especificados com o ponto de recursão apresentado anteriormente. Este funcionamento pode ser melhor observado na figura 3.12.

Figura 3.12 – Expansão recursiva do *workflow* por 2 níveis.

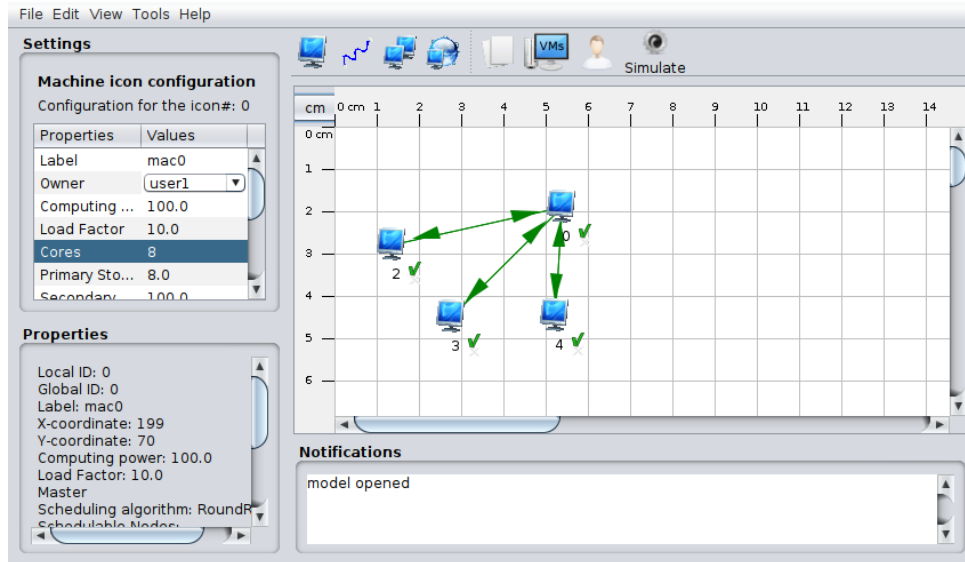
Fonte: Elaborado pelo autor.

3.2 Novas interfaces gráficas

Dados os ícones definidos, foram implementadas as novas interfaces gráficas no iSPD para modelagem do *workflow*. A partir da tela inicial do simulador, o usuário modela o *hardware*

e as máquinas virtuais, como representado na figura 3.13. Então deve clicar no ícone de configurar cargas () para modelar a carga de trabalho.

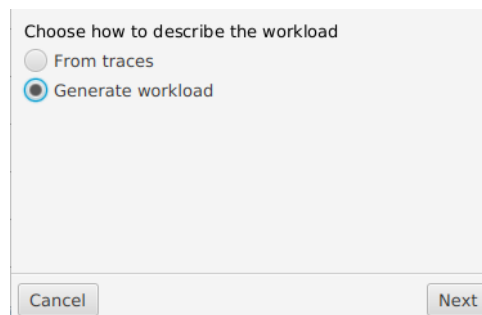
Figura 3.13 – Tela inicial do iSPD.



Fonte: Elaborado pelo autor.

Quando a janela de configuração de carga é exibida, o usuário escolhe entre cargas geradas por arquivos de *trace* ou gerada para cada nó mestre, como representado na figura 3.14.

Figura 3.14 – Janela de escolha da carga.



Fonte: Elaborado pelo autor.

Ao selecionar a carga obtida por *traces*, deve carregar um arquivo de *trace* previamente criado pelo iSPD. Já selecionado a opção de gerar a carga para cada nó mestre, então a janela da figura 3.15 é mostrada. Nela, há uma tabela com as entradas da cargas, que podem ser aleatórias ou em formato de *workflow*, incluindo o nó mestre a que se refere, o tempo médio de chegada da carga, e a especificação da carga.

Para as entradas aleatórias, o usuário insere o tamanho mínimo e máximo de computação e comunicação. Já para os *workflows*, o usuário gerencia seus *DAGs* interagindo com os botões

Figura 3.15 – Janela de geração de carga.

User	Scheduler	Quantity	Arrival ...	Type	Data
user1	mac0	1	2.5	dag	dag01

User:
☐ Random ☒ DAG

Scheduler:

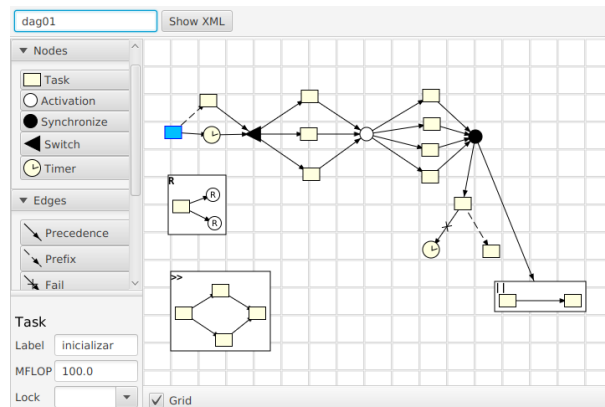
Quantity:

Arrival Time:

dag01
dag02
dag03

Fonte: Elaborado pelo autor.

de adição, edição e remoção na janela exposta na figura 3.16. Então na janela de edição do DAG, o usuário utiliza clica-e-arrasta para desenhar o *workflow* seguindo os ícones apresentados.

Figura 3.16 – Janela para edição de *workflow*.

Fonte: Elaborado pelo autor.

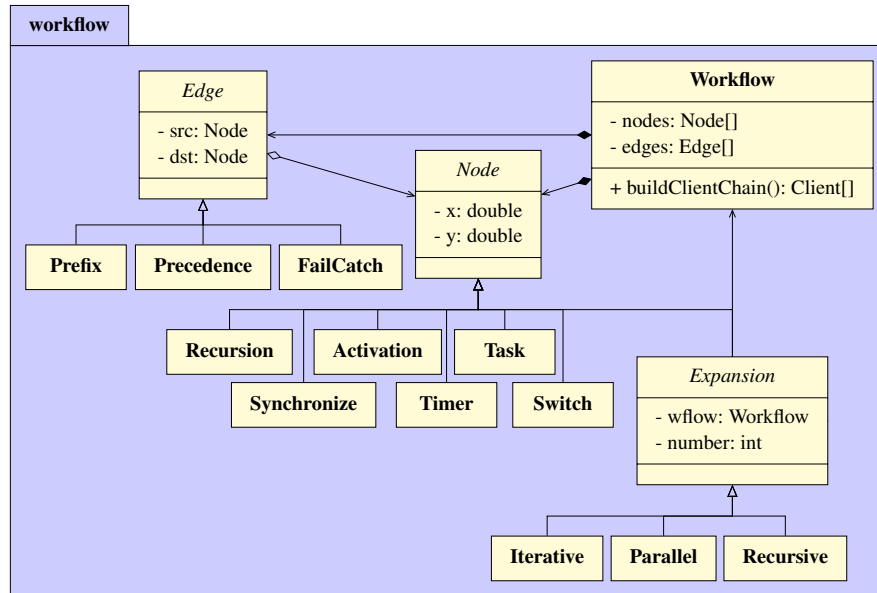
3.3 Arquitetura de implementação

3.3.1 Modelo Icônico

Para que o modelo icônico seja manipulado, o DAG que denota o *workflow* é salvo na memória como lista de nós e arestas, como representado no diagrama UML da figura 3.17. Como pode ser observado, as classes *Node* e *Edge* são abstratas e representam os nós e as arestas respectivamente. Os nós armazenam sua posição gráfica (x,y) e as arestas os nós que conecta. A

partir dessas classes abstratas, suas subclasses armazenam informações específicas referente à configuração de cada ícone. Por fim, a classe *Workflow* é composta de uma lista de nós e uma lista de arestas. Em particular, a classe *Expansion*, que também é um tipo de nó, contém uma estrutura interna da classe *Workflow* que será expandida.

Figura 3.17 – Diagrama UML do pacote que armazena o *workflow* na memória.



Fonte: Elaborado pelo autor.

3.3.2 Transformação da Cadeia de Clientes

A partir da organização estrutural do modelo gráfico, com a chamada da função *buildClientChain*, o mesmo é transformado em uma cadeia de clientes interpretáveis pelo modelo de filas do motor de simulação como eventos discretos. Este processo de transformação é descrito no Algoritmo 1.

Algorithm 1 Transformação do DAG em Cadeia de Clientes

- 1: Enquanto houver vértices não verificados no DAG
 - 2: Gera tarefa a partir de um vértice
 - 3: Para cada um de seus vértices vizinhos
 - 4: Encadeia a tarefa gerada adequadamente
 - 5: Retorna as tarefas iniciais da cadeia
-

No algoritmo, os vértices do grafo são percorridos de acordo com os seus tipos de carga e então um cliente do modelo de filas é gerado adequadamente para cada vértice. Essa adequação segue as regras na ordem listada:

- Caso o vértice seja um desvio probabilístico, um cliente cujo tempo de atendimento no sistema é nulo é gerado, então é escolhido um caminho de acordo com sua distribuição

de probabilidade, o que impede o algoritmo percorrer os vértices pertencentes aos outros caminho.

- Caso seja um ponto de sincronização ou ativação, o cliente gerado tem tempo de atendimento nulo no sistema, sendo relevante apenas suas dependências representadas pelas arestas que o conectam.
- Se o vértice é do tipo tarefa atômica, um cliente com suas características de tamanho computacional e *lock* é gerado. Quando tal cliente for processado por uma máquina do sistema de filas, seu tempo de atendimento é dado por $T \times P$, onde T é o tamanho computacional da tarefa e P é o poder de processamento da máquina.
- Se é uma espera temporizada, um cliente é gerado cujo tempo de atendimento no sistema equivale ao tempo de espera especificado.
- Se é uma das expansões, então o algoritmo é chamado recursivamente para o *workflow* interno o quão necessário for para expandir clientes referentes ao *workflow* interno no modelo final.
- Por fim, caso seja um ponto de recursão, que deve sempre estar no interior de uma expansão recursiva, um cliente equivalente ao do ponto de sincronização é gerado para ser encadeado com as novas cadeias do *workflow* interno da expansão.

No momento em que cada cliente é gerado, as arestas adjacentes ao respectivo vértice são verificadas e o cliente é encadeado com os clientes referentes aos vértices vizinhos. Finalmente, é retornado uma lista de clientes iniciais, isto é, aqueles que foram obtidos dos vértices sem predecessores.

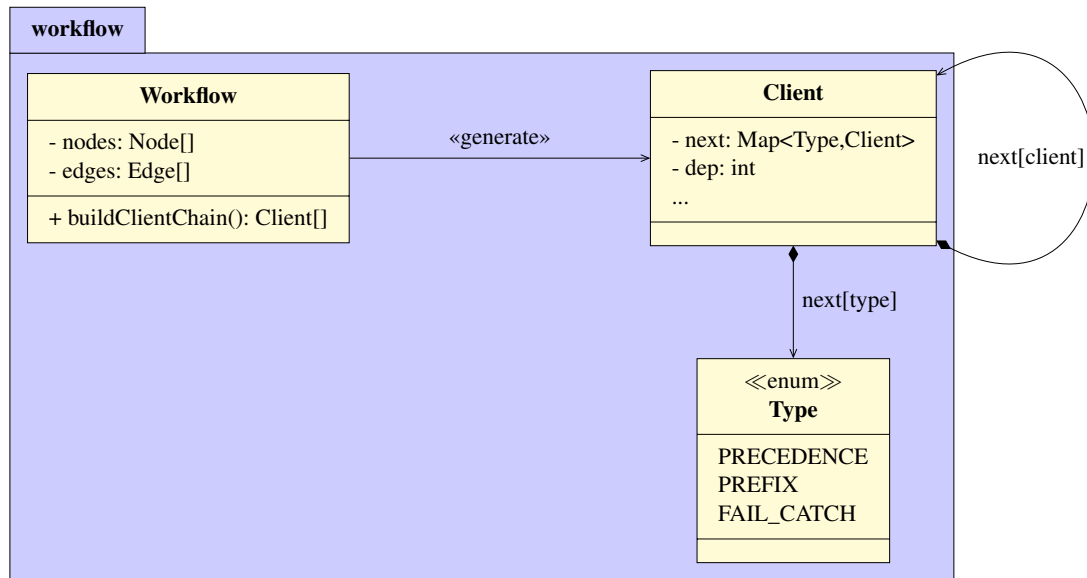
Ao final deste encadeamento, cada cliente contém uma lista de clientes dependentes denominada *next*, isto é, aqueles que ocorrem depois; e um inteiro *dep* que representa a quantidade de dependências que devem ser satisfeitas a priori. Esta estrutura está representada no diagrama UML 3.18.

3.3.3 Processamento pelo motor de simulação

A partir da cadeia de clientes formada e da rede de filas descrita pelo *hardware* modelado, a simulação é realizada de forma orientada a eventos discretos. Este processo pode ser dividido em duas partes: a preparação da fila de eventos discretos antes da simulação e a execução da simulação.

Na preparação, a fila de eventos discretos é preenchida com eventos de chegada dos clientes iniciais da cadeia de cliente, além de outros eventos relacionados a alocação de máquinas virtuais, que são descritos no trabalho de Silva (SILVA, 2015).

Figura 3.18 – Diagrama UML da Cadeia de tarefas.



Fonte: Elaborado pelo autor.

Durante a simulação, tais eventos são processados pelo motor de acordo com seu tipo, podendo ser chegada do cliente ao sistema, atendimento do cliente por um centro de serviço, falha durante o atendimento ou saída do sistema. Quando esses clientes são aqueles gerados no decorrer da transformação, uma das seguintes ações é tomada em relação as suas dependências:

- A ocorrência de um evento de chegada de uma tarefa no sistema não provoca criação de nenhum outro evento relacionado a outra tarefa que não seja si mesma.
- Na ocorrência de atendimento da tarefa t , suas tarefas sufixas são sinalizadas pela variável dep , isto é, para cada tarefa t_i do tipo sufixa em $t.next$, $t_i.dep := t_i.dep - 1$.
- Na ocorrência de falha de uma tarefa, realiza-se a mesma sinalização do item anterior para as tarefas do tipo tratamento de falha.
- Finalmente, na ocorrência de saída da tarefa, suas sucessoras são sinalizadas.

Durante esse processo, quando ocorre qualquer sinalização com variável dep , cada tarefa sinalizada é verificada e se estiver pronta, isto é $t.dep = 0$, um evento de chegada da mesma tarefa ao sistema é acrescentado à fila de eventos.

3.4 Considerações finais

Neste capítulo foi apresentada a abordagem gráfica para a modelagem dos *workflows* se utilizando de uma estrutura baseada no DAG que permite expansões as quais viabilizam a

modelagem de cargas mais complexas de forma gráfica, bem como que essa representação é armazenada na memória e transformada em uma estrutura de cadeia de clientes para que possa enfim ser processada pelo motor de simulação.

Nota-se que a partir dessa abordagem é possível cobrir boa parte das definições de *workflow* apresentadas na seção 2.2, como AND-split/join, OR-split, e OR-join com a utilização de ponto de sincronização, desvio probabilístico e ponto de ativação, respectivamente. Também foram realizadas abstrações das relações de precedência, prefixação e tratamento de falha por meio das arestas do grafo.

4 Testes

Para a validação deste trabalho foram realizados experimentos no simulador iSPD utilizando as novas funcionalidades de modelagem e simulação de *workflows* e coletados como resultados o tempo de execução (*make-span*) e então comparados com testes realizados em ambiente real no trabalho de Juve et al ([JUVE et al., 2013](#)).

Mais especificamente, foram escolhidos os workflows Epigenomics, SIPHT e Broadband, que são comumente utilizados em *benchmarks*. Para cada um desses workflows foram realizados 30 testes com as modelagens que serão descritas adiante, e coletados o tempo de execução simulado em cada teste.

O *hardware* utilizado nos testes em ambiente real está descrito na Tabela 4.1. Os *workflows* Epigenomics e Broadband foram executados em uma máquina virtual em nuvem de 8 núcleos, com o processador Xeon 2.33 GHz, com poder computacional estimado de 159 MFLOPS. O *workflow* SIPHT foi executado em um *cluster* de 13 nós de 8 núcleos com processador Xeon 3.16 GHz e poder computacional de 527 MFLOPS. Os poderem computacionais foram baseados na base de dados pública OpenBenchmark ([OPENBENCHMARK, 2010](#); [OPENBENCHMARK, 2014](#))

Tabela 4.1 – Ambiente utilizado para os testes em cada *workflow*.

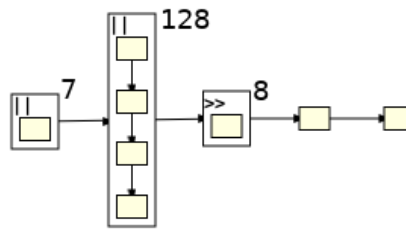
Workflow	Nós	Núcleos	Processador	Poder Computacional
SIPHT	13	8	Xeon (3.16 GHz)	527 MFLOPS
Epigenomics	1	8	Xeon (2.33 GHz)	159 MFLOPS
Broadband	1	8	Xeon (2.33 GHz)	159 MFLOPS

Fonte: Adaptado de Juve et al. ([JUVE et al., 2013](#))

A carga de cada tarefa do *workflow* foi estimada multiplicando-se o tempo médio de execução da tarefa no sistema real pelo poder de processamento da máquina que processa seu respectivo *workflow*.

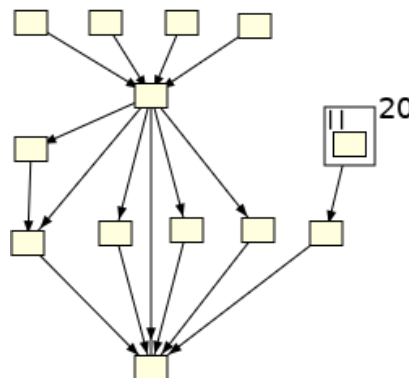
O *workflow* Epigenomics é composto de 7 tarefas iniciais paralelas de 5.445,9 MFlop, seguido de 128 blocos paralelos de 4 tarefas em sequência com 391,94 MFlop, 76,17 MFlop, 222,2 MFlop e 32.036 MFlop respectivamente. Ao término desses blocos paralelos, ocorrem 8 iterações de uma tarefa de 1.747,07 MFlop, que precede outras duas tarefas em sequência de 6.913,7 MFlop e 8.878,2 MFlop, respectivamente. Sua modelagem na interface do iSPD está representada na Figura 4.1.

O SIPHT se inicia com 4 tarefas paralelas, de 3.471,7 MFlop, 627.279 MFlop, 27.086 MFlop e 3.491,1 GFlop. Após a conclusão dessas tarefas ocorre seu *join* por uma tarefa de 13.116 MFlop. Após essa são liberadas outras 5 tarefas de 769,7 MFlop, 3.553 MFlop, 632,6

Figura 4.1 – Configuração do *workflow* Epigenomics.

Fonte: Elaborado pelo autor.

MFlop, 464.845 MFlop e 717 MFlop, respectivamente, sendo que a primeira também precede a segunda. Paralelamente a este processo, ocorrem 20 tarefas paralelas de 1.012 MFlop, as quais passam por um *join* com uma tarefa de 31,63 MFlop. Por fim, a partir da conclusão das últimas 5 tarefas independentes entre si, como é possível observar na Figura 4.2, é executada uma última tarefa, de 147,6 MFlop.

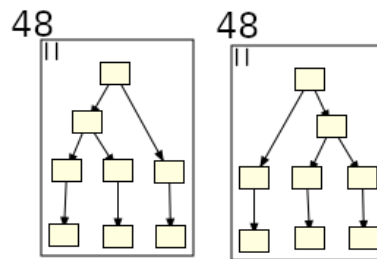
Figura 4.2 – Configuração do *workflow* Sipht.

Fonte: Elaborado pelo autor.

Já o *workflow* Broadband consiste em dois grupos similares de *workflows*, cada um com 48 ocorrências em paralelo. O primeiro grupo se inicia com uma tarefa de 90,63 MFlop seguida de duas tarefas, uma de 10.862,88 MFlop e outra de 13.502,28 MFlop, a qual precede mais duas tarefas de 486,54 MFlop e 31.227,6 MFlop, respectivamente. Por fim as três tarefas mais a baixo na hierarquia da árvore precedem três tarefas iguais de 54,06 MFlop. Já o segundo grupo de workflow tem característica similar ao primeiro, tendo como única exceção a primeira tarefa sendo de 68,37 MFlop. Essa caracterização é ilustrada na Figura 4.3.

Os resultados estão descritos na tabela 4.2. Para cada *workflow* é considerado o tempo de execução real obtido do trabalho e a média e o desvio padrão do tempo do mesmo *workflow* simulado no iSPD para dois casos diferentes:

- O caso denominado **Workflow** se trata da modelagem exibida nas figuras 4.1, 4.2 e 4.1 que incluem as dependências entre as tarefas.

Figura 4.3 – Configuração do *workflow* Broadband.

Fonte: Elaborado pelo autor.

- Já o caso denominado **Independente** descreve o mesmo modelo, mas desconsiderando suas dependências, isto é, com todas as tarefas podendo ser executadas simultaneamente.

Tabela 4.2 – Resultados dos experimentos (tempos em segundos)

Workflow	Real	Workflow			Independentes		
		Simulado	Desvio	Erro	Simulado	Desvio	Erro
SIPHT	4.253	3.842,65	199,18	09,66%	3.343,27	215,45	21,39%
Epigenomics	4.060	3.632,24	33,39	10,54%	3434,56	38,97	15,40%
Broadband	4.803	4.555,58	37,96	5,15%	4.513,79	31,99	6,02%

Fonte: Elaborado pelo autor.

Dos resultados, pode-se destacar que de forma geral o tempo de execução de cada um dos experimentos considerando as dependências foram maiores em relação àqueles em que as tarefas foram consideradas totalmente independentes, o que mostra que o motor de simulação está de fato considerando as restrições de execução que o *workflow* impõe, o que é esperado.

Analisando cada caso em particular, observa-se que a melhora apresentada no teste do *workflow* SIPHT foi a mais significativa, em seguida do Epigenomics e do Broadband, sendo a melhora menos significativa. Essa mudança de comportamento pode estar associada tanto a própria caracterização da carga de trabalho, uma vez que cargas com maior número de tarefas paralelizáveis entre si são menos influenciadas pela adição das dependências, quanto ao hardware que foi utilizado para o desenvolvimento, pois o ganho de representatividade do simulador fica mais visível quando há mais máquinas para realizar o processamento em paralelo.

5 Conclusão

Neste trabalho foram apresentados estudos envolvendo computação em nuvem e *workflows*, simulação orientada a eventos discretos e modelagem utilizando-se teoria das filas, modelos gráficos para a representação de *workflows*, e simuladores de nuvem da literatura. Então foi apresentada uma abordagem gráfica para a modelagem de *workflows* baseada em DAG juntamente com a descrição de sua implementação no simulador iSPD. Por fim o projeto foi validado com testes no simulador confrontados com testes em ambiente real.

Da proposição da abordagem gráfica, concluí-se que foi possível desenvolver uma forma gráfica da modelagem de *workflow* que visa facilitar a modelagem para o usuário final uma vez que não é necessário conhecimento de linguagem de programação ou marcação.

Outro ponto interessante é que a utilização das expansões possibilitaram a modelagem de *workflows* consideravelmente maiores com poucos ícones, como pôde ser observado na modelagem do *workflow* SIPHT por exemplo em que foram utilizados 8 tarefas e 3 expansões para representar um *workflow* de 529 tarefas ao todo.

A partir dos experimentos, concluí-se que houve melhora de representatividade dos testes reais, quando observado os resultados da simulação com especificação de *workflow* em relação aos resultados de cargas independentes.

5.1 Trabalhos Futuros

Alguns exemplos de trabalhos futuros são a implementação e testagem de escalonadores e alocadores de máquinas virtuais cientes de dependências.

Além disso, o conceito de expansões apresentado neste trabalho para a modelagem da carga de trabalho pode ser utilizado para outras modelagens que envolvem grafos, como a modelagem do hardware dentro do iSPD.

Por fim, os modelos gráficos dos *workflows* poderiam ser utilizados para gerar templates de gerenciadores de *workflows* reais como arquivos DAX, para o Pegasus, arquivos de configuração em python para o Apache Airflow, arquivo de configuração JSON para ser utilizado na Amazon Simple Workflow Service ou em outros serviços de *workflow* em nuvem.

Referências

- AALST, W. V. D.; HEE, K. M. V.; HEE, K. van. *Workflow management: models, methods, and systems*. [S.l.]: MIT press, 2004.
- AIRFLOW, A. *Apache Airflow*. 2020. Disponível em: <<https://airflow.apache.org/>>.
- AMAZON. *Amazon Elastic Computer Service*. 2020. Disponível em: <<https://aws.amazon.com/pt/ec2/>>.
- ARENALES, M.; ARMENTANO, V. et al. *Pesquisa operacional*. [S.l.]: Elsevier Brasil, 2006.
- BALDISSERA, L. V. et al. Modelagem e simulação de relações entre tarefas computacionais em ambiente de nuvem. In: SBC. *Anais da XI Escola Regional de Alto Desempenho de São Paulo*. [S.l.], 2020. p. 18–21.
- BINU, A.; KUMAR, G. S. Virtualization techniques: a methodical review of xen and kvm. In: SPRINGER. *International Conference on Advances in Computing and Communications*. [S.l.], 2011. p. 399–410.
- BOLOGNESI, T.; BRINKSMA, E. Introduction to the iso specification language lotos. *Computer Networks and ISDN systems*, Elsevier, v. 14, n. 1, p. 25–59, 1987.
- BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. *Cloud computing: Principles and paradigms*. [S.l.]: John Wiley & Sons, 2010. v. 87.
- CALHEIROS, R. N. et al. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, Wiley Online Library, v. 41, n. 1, p. 23–50, 2011.
- CALZAROSSA, M. C. et al. Workloads in the clouds. In: *Principles of Performance and Reliability Modeling and Evaluation*. [S.l.]: Springer, 2016. p. 525–550.
- CHEN, W.; DEELMAN, E. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In: IEEE. *2012 IEEE 8th international conference on E-science*. [S.l.], 2012. p. 1–8.
- CHEN, W. et al. Balanced task clustering in scientific workflows. In: IEEE. *2013 IEEE 9th International Conference on e-Science*. [S.l.], 2013. p. 188–195.
- DAVIDSSON, P. Multi agent based simulation: beyond social simulation. In: SPRINGER. *International workshop on multi-agent systems and agent-based simulation*. [S.l.], 2000. p. 97–107.
- DEELMAN, E. et al. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, Elsevier, v. 46, p. 17–35, 2015.
- ELWASIF, W. R. et al. A dependency-driven formulation of parareal: parallel-in-time solution of pdes as a many-task application. In: ACM. *Proceedings of the 2011 ACM international workshop on Many task computing on grids and supercomputers*. [S.l.], 2011. p. 15–24.
- GOOGLE. *App Engine*. 2020. Disponível em: <<https://cloud.google.com/appengine>>.

GOOGLE. *Compute Engine*. 2020. Disponível em: <<https://cloud.google.com/compute>>.

JUVE, G. et al. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, Elsevier, v. 29, n. 3, p. 682–692, 2013.

KEELE, S. et al. *Guidelines for performing systematic literature reviews in software engineering*. [S.l.], 2007.

KIM, K.-H. Workflow dependency analysis and its implications on distributed workflow systems. In: IEEE. *17th International Conference on Advanced Information Networking and Applications*, 2003. AINA 2003. [S.l.], 2003. p. 677–682.

KLIAZOVICH, D.; BOUVRY, P.; KHAN, S. U. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, Springer, v. 62, n. 3, p. 1263–1283, 2012.

LEEMIS, L. M.; PARK, S. K. *Discrete-event simulation: A first course*. [S.l.]: Pearson Prentice Hall Upper Saddle River, NJ, 2006.

LIU, R.; KUMAR, A. An analysis and taxonomy of unstructured workflows. In: SPRINGER. *International Conference on Business Process Management*. [S.l.], 2005. p. 268–284.

LOUIS, G. B.; GILBERT, A. *Modelling and simulation: Exploring dynamic system behaviour*. Ottawa: School of information technology and engineering, 2007.

MANACERO, A. et al. iSPD: an iconic-based modeling simulator for distributed grids. In: SOCIETY FOR COMPUTER SIMULATION INTERNATIONAL. *Proceedings of the 45th Annual Simulation Symposium*. [S.l.], 2012. p. 5.

MARZAKIS, M.; NIKOLAOU, C. Towards adaptive scheduling of tasks in transactional workflows. In: IEEE. *Winter Simulation Conference Proceedings, 1995*. [S.l.], 1995. p. 604–611.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National ... , 2011.

MICROSOFT. *IaaS (infraestrutura como serviço) do Azure*. 2020. Disponível em: <<https://azure.microsoft.com/pt-br/overview/what-is-azure/iaas/>>.

MILOJIČIĆ, D.; LLORENTE, I. M.; MONTERO, R. S. Opennebula: A cloud management tool. *IEEE Internet Computing*, IEEE, v. 15, n. 2, p. 11–14, 2011.

MOONEY, C. Z. *Monte carlo simulation*. [S.l.]: Sage publications, 1997. v. 116.

NS. *User Information - nsnam*. 2020. Disponível em: <http://nsnam.sourceforge.net/wiki/index.php/User_Information>.

NÚÑEZ, A. et al. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, Springer, v. 10, n. 1, p. 185–209, 2012.

OPENBENCHMARK. *Himeno Benchmark*. 2010. Disponível em: <<https://openbenchmarking.org/result/1012296-IV-AMAZONC1X75>>. Acesso em: 26 jun. 2020.

OPENBENCHMARK. *SciMark - Computational Test: Composite*. 2014. Disponível em: <<https://openbenchmarking.org/result/1408019-KH-XW8600SCI06>>. Acesso em: 26 jun. 2020.

- SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, Published by Foundation of Computer Science, v. 55, n. 3, p. 38–42, 2012.
- SILVA, D. T. d. Abordagem icônica para modelagem e simulação de ambientes de computação em nuvem? Universidade Estadual Paulista (UNESP), 2015.
- SILVA, D. T. d. et al. Simulação de sistemas de computação em nuvem para o ispd. *Interciência & Sociedade*, p. 86–93, 2015.
- SOARES, L. F. G. *Modelagem e simulação discreta de sistemas*. [S.l.]: Campus, 1992.
- TICK, J. Workflow model representation concepts. In: *Proceedings of 7th International Symposium of Hungarian Researchers on Computational Intelligence, HUCI*. [S.l.: s.n.], 2006. p. 24–25.
- TIGHE, M. et al. Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management. In: IEEE. *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualization management (svm)*. [S.l.], 2012. p. 385–392.
- VARGA, A. Omnet++ discrete event simulation system: User manual. In: *Proc. of the European Simulation Multiconference (ESM'2001)*. [S.l.: s.n.], 2001. p. 1–7.
- WALKER, D. W.; DONGARRA, J. J. Mpi: a standard message passing interface. *Supercomputer*, ASFRA BV, v. 12, p. 56–68, 1996.
- WENG, N. et al. Mapping task graphs onto network processors using genetic algorithm. In: IEEE. *2008 IEEE/ACS International Conference on Computer Systems and Applications*. [S.l.], 2008. p. 481–488.
- WICKREMASINGHE, B.; CALHEIROS, R. N.; BUYYA, R. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: IEEE. *2010 24th IEEE international conference on advanced information networking and applications*. [S.l.], 2010. p. 446–452.
- YU, J.; BUYYA, R. A taxonomy of scientific workflow systems for grid computing. *ACM Sigmod Record*, ACM New York, NY, USA, v. 34, n. 3, p. 44–49, 2005.

A Microdados dos experimentos realizados

Tabela A.1 – Microdados dos testes realizados. Tempo em segundos.

#	SIPHT		Epigenomics		Broadband	
	Workflow	Independente	Workflow	Independente	Workflow	Independente
1	3.737,45	3.434,76	3.604,93	3.460,33	4.539,18	4.449,56
2	3.505,89	3.044,93	3.677,24	3.397,06	4.473,25	4.527,02
3	3.822,43	3.099,76	3.686,6	3.419,18	4.539,69	4.470,85
4	4.103,2	3.345,65	3.693,07	3.492,74	4.565,87	4.523,26
5	3.729,54	3.059,67	3.596,48	3.410,17	4.587,97	4.528,24
6	4.001,05	3.144,81	3.588,73	3.361,78	4.528,35	4.509,09
7	3.909,09	3.002,93	3.593,66	3.491,9	4.604,85	4.490,52
8	3.633,28	3.601,3	3.645,29	3.402,76	4.553,7	4.521,43
9	3.801,73	3.192,45	3.646,99	3.350,27	4.558,62	4.514,12
10	3.616,64	3.607,88	3.656,25	3.440,54	4.527,7	4.461,82
11	3.644,86	3.498,11	3.650,68	3.429,02	4.521,89	4.459,74
12	4.052,71	3.602,83	3.624,7	3.431,72	4.556,23	4.539,44
13	4.091,78	3.130,71	3.650,61	3.492,15	4.539,79	4.546,87
14	3.717,59	3.331,18	3.605,89	3.442,78	4.517,09	4.484,11
15	3.860,65	3.092,22	3.611,96	3.430	4.517,32	4.548,32
16	3.787,88	3.315,48	3.695,05	3.445,61	4.585,21	4.536,55
17	3.471,08	3.236,77	3.617,93	3.444,28	4.588,61	4.542,07
18	3.693,75	3.373,73	3.579,14	3.460,32	4.651,53	4.527,62
19	4.106,35	3.120,13	3.644,42	3.462,47	4.610,58	4.479,09
20	3.917,51	3.007,85	3.652,87	3.427,87	4.592,76	4.512,87
21	3.704,17	3.599,36	3.615,27	3.476	4.560,14	4.512,88
22	4.062,61	3.608,7	3.604	3.418,57	4.507,5	4.534,5
23	3.675,86	3.243,67	3.692,26	3.396,87	4.530,25	4.506,24
24	4.141,65	3.550,92	3.627,59	3.473,95	4.576,01	4.488,35
25	4.032,57	3.588,38	3.585,06	3.379,05	4.522,17	4.577,81
26	4.023,88	3.643,36	3.619,21	3.394,59	4.565,44	4.520,55
27	3.925,75	3.522,92	3.651,86	3.401,52	4.578,95	4.559,61
28	4.199,15	3.629,92	3.631,5	3.421,49	4.538,83	4.453,05
29	3.670,87	3.256,54	3.623,64	3.483,75	4.572,99	4.495,45
30	3.638,64	3.411,23	3.594,3	3.498,02	4.567,68	4.517,05