CAIO RAFAEL DO NASCIMENTO SANTIAGO

Desenvolvimento de um ambiente de computação voluntária baseado em computação ponto-a-ponto

São Paulo

2015

CAIO RAFAEL DO NASCIMENTO SANTIAGO

Desenvolvimento de um ambiente de computação voluntária baseado em computação ponto-a-ponto

Versão original.

Dissertação apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação.

Área de Concentração: Sistemas de Informação.

Orientador: Prof. Dr. Luciano Antonio Digiampietri

São Paulo
2015

CATALOGAÇÃO-NA-PUBLICAÇÃO
Universidade de São Paulo. Escola de Artes, Ciências e Humanidades. Biblioteca

Santiago, Caio Rafael do Nascimento
Desenvolvimento de um ambiente de computação voluntária baseado em computação ponto-a-ponto / Caio Rafael do Nascimento Santiago ; orientador, Luciano Antonio Digiampietri. – São Paulo, 2015
66 f. : il.

Dissertação (Mestrado em Ciências) - Programa de Pós-Graduação em Sistemas de Informação, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo
Versão original

1. Sistemas colaborativos. 2. Redes de computadores. 3. Sistemas distribuídos. 4. Pesquisa científica. I. Digiampietri, Luciano Antonio, orient. II. Título.

CDD 22.ed. – 004.65

Dissertação de autoria de Caio Rafael do Nascimento Santiago, sob o título *"Desenvolvimento de um ambiente de computação voluntária baseado em computação ponto-a-ponto"*, apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação, na área de concentração Sistemas de Informação, aprovada em ____ de _____ de _____ pela comissão julgadora constituída pelos doutores:

Prof. Dr. _____
Presidente
Instituição: _____

Prof. Dr. _____
Instituição: _____

Prof. Dr. _____
Instituição: _____

Dedico este trabalho a minha mãe por todo o apoio e compreensão, sem os quais não seria possível alcançar o que alcancei.

AGRADECIMENTOS

Agradeço antes de mais nada a minha mãe Sandra por todo o esforço, dedicação, e principalmente pela compreensão que foram fundamentais para enfrentar os desafios que me trouxeram até aqui.

Ao meu orientador, Prof. Dr. Luciano Antonio Digiampietri, por ter me guiado no decorrer deste trabalho, e sem o qual nada disso seria possível.

A minha namorada Karen por todos os momentos de felicidade que fizeram esta jornada não parecer tão longa.

Aos bons amigos que a tantos anos me acompanham, sejam eles na faculdade ou fora dela, por todos os momentos de descontração que dão folego para trilhar essa jornada.

A todos os professores da EACH-USP da graduação e da pós-graduação pela contribuição na minha formação.

A Fapesp pelo apoio financeiro (projetos 2013/07935-5 e 2009/10413-5).

"O aspecto mais triste da vida de hoje é que a ciência ganha em conhecimento mais rapidamente que a sociedade em sabedoria."

(Isaac Asimov)

RESUMO

SANTIAGO, Caio Rafael do Nascimento. **Desenvolvimento de um ambiente de computação voluntária baseado em computação ponto-a-ponto**. 2015. 66 f. Dissertação (Mestrado em Ciências) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2015.

As necessidades computacionais de experimentos científicos muitas vezes exigem computadores potentes. Uma forma alternativa de obter esse processamento é aproveitar o processamento ocioso de computadores pessoais de modo voluntário. Essa técnica é conhecida como computação voluntária e possui grande potencial na ajuda aos cientistas. No entanto existem diversos fatores que podem reduzir sua eficiência quando aplicada a experimentos científicos complexos, por exemplo, aqueles que envolvem processamento de longa duração, uso de dados de entrada ou saída muito grandes, etc. Na tentativa de solucionar alguns desses problemas surgiram abordagens que aplicam conceitos de computação ponto-a-ponto. Neste projeto foram especificados, desenvolvidos e testados um ambiente e um escalonador de atividades que aplicam conceitos de computação ponto-a-ponto à execução de workflows com computação voluntária.

Palavras-chave: Computação Voluntária; Desktop Grid; Ponto-a-Ponto; Escalonamento

ABSTRACT

SANTIAGO, Caio Rafael do Nascimento. **Development of an volunteer computing environment based in peer-to-peer computing**. 2015. 66 p. Dissertation (Master of Science) – School of Arts, Sciences and Humanities, University of São Paulo, São Paulo, 2015.

The scientific experiments computational needs often require powerful computers. One alternative way to obtain this processing power is taking advantage of the idle processing of personal computers as volunteers. This technique is known as volunteer computing and has great potential in helping scientists. However, there are several issues which can reduce the efficiency of this approach when applied to complex scientific experiments, such as, the ones with long processing time, very large input or output data, etc. In an attempt to solve these problems some approaches based on P2P concepts arisen. In this project a workflow execution environment and a scheduler of activities were specified, developed and tested applying P2P concepts in the workflows execution using volunteer computing.

Keywords: Volunteer Computing; Desktop Grid; Peer-to-Peer; Scheduling

LISTA DE FIGURAS

Figura 1 – Estruturas básicas de workflows 21
Figura 2 – Arquitetura do projeto Jovem Pesquisador 23
Figura 3 – Diagrama de classes da estrutura de representação do workflow 34
Figura 4 – Diagrama de classes da estrutura modificada de representação do workflow 34
Figura 5 – Diagrama de classes da estrutura do executor Workflow 35
Figura 6 – Diagrama de classes da estrutura modificada do executor Workflow . . 36
Figura 7 – Diagrama de classes da estrutura de comunicação entre o servidor e os voluntários 36
Figura 8 – Workflow de teste baseado em um problema real de análise de redes sociais 44
Figura 9 – Utilização do processamento dos testes baseados em um problema real de análise de redes sociais 50
Figura 10 – Processamento em função do tempo (segundos) de cada voluntário no caso de exemplos simplificados (com uma estrutura) 52

LISTA DE TABELAS

Tabela 1 – Dados extraídos dos artigos referentes à estrutura de dados 25
Tabela 2 – Dados extraídos dos artigos referentes à avaliação do escalonamento . . 29
Tabela 3 – Configurações do computadores utilizados 42
Tabela 4 – Tempo de duração do processamento dos casos de testes reais de análise de redes sociais 47
Tabela 5 – Tempo médio de espera transferindo dados nos voluntários nos casos de teste de análise de redes sociais 48
Tabela 6 – Dados transferidos por cada voluntário em MBs no caso real de redes sociais 48
Tabela 7 – Dados transferidos por cada voluntário em MBs no caso real de redes sociais sem a execução de atividades no servidor (com redundância) . . 49
Tabela 8 – Tempo de duração do processamento dos casos de testes exemplos simplificados (com uma estrutura) 51
Tabela 9 – Dados transferidos por cada voluntário em MBs no caso de exemplos simplificados (com uma estrutura) 53
Tabela 10 – Tempo de duração do processamento dos casos de testes exemplos simplificados (com 10 estruturas) 53
Tabela 11 – Comparação entre o tempo de CV com o tempo da CV com P2P (com 10 estruturas) 54
Tabela 12 – Tempo médio de espera transferindo dados pelos voluntários nos casos de exemplos simplificados (com 10 estruturas) 54
Tabela 13 – Dados transferidos por cada voluntário em MBs no caso de exemplos simplificados (com 10 estruturas) 55
Tabela 14 – Tempo de duração do processamento dos casos de testes exemplos simplificados (com 10 estruturas e redundância) 55
Tabela 15 – Comparação entre o tempo de CV com o tempo da CV com P2P (com e sem redundância – 10 estruturas) 56
Tabela 16 – Dados transferidos por cada voluntário em MBs no caso de exemplos simplificados (com 10 estruturas e redundância) 56

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Objetivos 14
1.2 Metodologia 14
1.3 Organização do Texto 15

2 CONCEITOS FUNDAMENTAIS 17
2.1 Workflows Científicos 17
2.2 Computação Voluntária 18
2.3 Desktop Grids 19
2.4 Composição de Serviços 19
2.5 Escalonamento com ponto-a-ponto 20
2.6 Benchmark 20
2.7 Infraestrutura 21
2.8 Considerações Finais 22

3 REVISÃO SISTEMÁTICA 24
3.1 Resumo da Revisão 24
3.1.1 Estrutura de dados 24
3.1.2 Algoritmo de escalonamento 27
3.1.3 Avaliação do escalonamento 28
3.2 Comparação com a Solução Desenvolvida no Mestrado 31
3.3 Considerações Finais 32

4 SOLUÇÃO DESENVOLVIDA 33
4.1 Extensão do Modelo de Representação de Workflows 33
4.2 Comunicação entre Computadores 36
4.3 Computação Voluntária aplicando técnicas de Ponto-a-Ponto 38
4.4 Considerações Finais 40

5 AVALIAÇÃO DA SOLUÇÃO E RESULTADOS 42
5.1 Avaliação da Solução 42

5.1.1 Análise de Redes Sociais 43
5.1.2 Exemplos Simplificados 45
5.2 Resultados 46
5.2.1 Análise de Rede Sociais 46
5.2.2 Exemplos Simplificados 49
5.3 Considerações Finais 56

6 CONCLUSÕES 58
6.1 Principais Contribuições 59
6.2 Trabalhos Futuros 60

REFERÊNCIAS 63

1 INTRODUÇÃO

As necessidades computacionais de experimentos científicos muitas vezes exigem computadores potentes, que em geral são caros e que, potencialmente, ficarão ociosos em boa parte do tempo (MUTKA; LIVNY, 1991; ACHARYA; EDJLALI; SALTZ, 1997). Por outro lado, o avanço dos computadores pessoais, com CPUs multi-núcleos e GPUs, tipicamente atendem às necessidades de seus usuários com facilidade. Portanto temos uma situação onde computadores pessoais passam a maior parte do tempo ociosos e computadores científicos sobrecarregados durante períodos de pico de experimentos.

Experimentos científicos muitas vezes são modelados como workflows científicos (MEDEIROS et al., 1996), que por si só já possuem uma grande capacidade de paralelização. Em geral, um workflow científico é executado em um único computador, que não é necessariamente muito potente, ou em um *grid*, que exige recursos e planejamento. Uma recente abordagem para aproveitar o poder dos workflows é utilizar vários computadores pessoais, como por exemplo *desktop grids* (KONDO et al., 2007; ANDERSON, 2004) ou computação voluntária (ANDERSON; FEDAK, 2006).

A computação voluntária (CV) tenta aproveitar o recurso ocioso "doando-o" para um determinado projeto, isto é, o responsável pelo projeto transmite (geralmente via internet) para outro computador (doador) uma atividade para que este a processe. Esse paradigma tem o poder de fornecer muito processamento (ANDERSON, 2004), mas quando se trata de experimentos científicos existem muitos fatores que podem tornar esta solução pouco eficiente, como por exemplo, processamento de longa duração (DETHIER et al., 2008), grande volume de dados (DUAN et al., 2012), heterogenia de sistemas (DORNEMANN et al., 2012) e instabilidade dos computadores voluntários (DIAS et al., 2010).

No intuito de resolver algumas destas questões surgiram propostas que empregam conceitos de computação *ponto-a-ponto* (P2P) (MAJITHIA et al., 2004; ZHOU; LO, 2006; ZHAO; YANG; XU, 2009) em workflows que utilizam processamento voluntário. Esta é uma solução bastante dinâmica e que, portanto, é capaz de lidar com ambientes heterogêneos e ser tolerante a falhas, além de minimizar ou ao menos descentralizar o tráfego de dados. Pode-se encarar esse conceito como uma extensão da coreografia de serviços (PELTZ, 2003).

A arquitetura tradicional de computação voluntária se assemelha a execução tradicional de serviços web (*web services*). A composição de forma descentralizada de serviços web, na qual os serviços se comunicam diretamente entre si (*ponto-a-ponto*) possibilitou melhorias significativas em termos de escalabilidade e transferência de dados (DUAN et al., 2012).

Este projeto visa a ajudar a enfrentar os desafios de escalonar tarefas em workflows científicos em uma arquitetura descentralizada P2P utilizando-se computação voluntária.

1.1 Objetivos

O objetivo geral deste projeto é verificar se a aplicação de técnicas ponto-a-ponto (P2P) em ambientes de computação voluntária (CV) pode acarretar em uma melhora no desempenho, quando comparada com a computação voluntária tradicional.

Para alcançar esse objetivo os seguintes objetivos específicos foram estabelecidos:

- Construir/expandir um Sistema de Gerenciamento de Workflows (SGW) de fácil modularização e expansão, com o intuito de desenvolver diversos ambientes de execução de workflows capazes de trabalhar com diferentes algoritmos de escalonamento. O SGW resultante deve ser o mais transparente possível para o projetista do workflow, não sendo necessário conhecimento sobre os computadores participantes do sistema e nem sobre a estrutura de rede na qual se comunicam;
- Desenvolver um ambiente de computação voluntária;
- Especificar e desenvolver um ambiente de computação voluntária que aplique técnicas P2P;
- Avaliar o desempenho do ambiente proposto quando comparado com a computação voluntária tradicional e com a execução local de workflows.

1.2 Metodologia

A metodologia deste trabalho foi iniciada com a realização de um estudo teórico a fim de verificar o que já havia sido feito na literatura correlata e estudar a viabilidade deste trabalho. Para este fim foi feita uma revisão sistemática para englobar o principal tópico deste trabalho: algoritmos de escalonamento em workflows científicos que apliquem técnicas

de P2P. A revisão indicou que a literatura correlata se mostrou escassa, corroborando com a contribuição potencial deste projeto.

Em seguida foi estendido um Sistema de Gerenciamento de Workflows (SGW) de forma a possibilitar a construção de estruturas para o escalonamento de workflows que fosse flexível o suficiente para utilizar diferentes tipos de técnicas de escalonamento. Para isto, este trabalho utilizou e expandiu o SGW desenvolvido no projeto Jovem Pesquisador em Centros Emergentes – FAPESP do orientador, intitulado "Gerenciamento e Composição Automática de Serviços Web Semânticos" (DIGIAMPIETRI et al., 2011; DIGIAMPIETRI et al., 2013; ZUNIGA et al., 2014).

A partir do SGW existente foram desenvolvidos três métodos de escalonamento de workflows. O primeiro método foi um método de execução local, envolvendo apenas um computador, baseado no Método do Caminho Crítico, no qual o workflow é visto como um dígrafo e prioriza-se o caminho mais longo. O segundo método é a computação voluntária tradicional, onde não há a interação entre computadores voluntários. O terceiro é um método de computação voluntária aplicando técnicas de computação P2P, e é de fato o principal objetivo deste trabalho. Os dois primeiros métodos têm duas funções neste trabalho: servem como base para o desenvolvido do terceiro método, pois seus códigos são parcialmente reutilizados; e contribuem para a avaliação do desempenho do terceiro método.

A avaliação do desempenho da solução desenvolvida foi realizada pela comparação com a computação voluntária tradicional e a execução local, a partir de diferentes métricas aplicadas aos resultados obtidos em dois cenários de testes. No primeiro cenário são utilizados exemplos simplificados (*toy examples*) baseados nas estruturas básicas de workflow (BHARATHI et al., 2008). Neste cenário, cada estrutura é confrontada com situações de alta transferência dados e processamento de longa duração. O segundo cenário é um caso de teste real da área de análise de redes sociais.

1.3 Organização do Texto

Esta dissertação está dividida em seis capítulos, incluindo esta introdução. Os demais capítulos estão organizados da seguinte forma:

- O Capítulo 2 contém um resumo de alguns conceitos teóricos pertinentes para o entendimento deste trabalho. Estes conceitos envolvem principalmente a computação voluntária, os elementos da organização de workflows e os conceitos de composição de serviços que inspiraram a solução desenvolvida.
- O Capítulo 3 sumariza a revisão sistemática realizada abordando soluções similares, isto é, sistema de computação voluntária com elementos presentes de P2P.
- O Capítulo 4 descreve a solução especificada e desenvolvida, detalhando tanto os aspectos do servidor quanto dos voluntários, bem como descrevendo a forma como eles se comunicam. Além disso, também é descrito como foi estruturado e estendido o SGW que serviu de base para o resto do desenvolvimento. Por fim, esse capítulo também apresenta os outros métodos de escalonamento (utilizados para comparação de desempenho), com um enfoque principalmente na organização dos computadores.
- O Capítulo 5 descreve como a solução foi avaliada, isto é, como os métodos de escalonamento foram comparados, os cenários de testes e as formas de execução. Além disso, neste capítulo os resultados destas comparações são apresentados e discutidos a partir de algumas métricas.
- Finalmente, o Capítulo 6 apresenta as considerações finais sobre este trabalho, onde são expostas as contribuições alcançadas, assim como possíveis melhorias e trabalhos futuros.

2 CONCEITOS FUNDAMENTAIS

Neste capítulo são descritos os conceitos fundamentais acerca do trabalho, os quais serão utilizados como base ao longo desta dissertação. Estes conceitos envolvem principalmente a computação voluntária, os elementos da organização de workflows e os conceitos de composição de serviços que inspiraram a solução desenvolvida.

2.1 Workflows Científicos

Workflows (ou fluxos de trabalho) podem ser definidos como modelos de representação e execução de um conjunto de atividades a serem executadas de forma coordenada. Essa estrutura concede um alto poder de representação e paralelização, portanto são muito úteis e amplamente utilizados, especialmente, em sistemas distribuídos. Os workflows podem variar muito em forma, espécie ou propósito, e uma das diversas maneiras de classifica-los é a divisão entre workflows corporativos (ou de negócios) e científicos.

A natureza dos workflows científicos é bastante diferente dos workflows corporativos (MEDEIROS et al., 1996; TAYLOR et al., 2007). Tipicamente, workflows corporativos se dedicam mais a armazenar objetivos e manipulá-los conforme regras predefinidas, enquanto que workflows científicos enfocam na transferência e manipulação de dados. O volume de dados costuma ser significativamente maior nos workflows científicos e, embora ambos possuam atividades de longa duração, nos workflows corporativos o tempo costuma ser determinado pela interação dos participantes com o workflow, diferente dos workflows científicos em que o tempo é determinado pelo processamento pesado das atividades que o compõem.

Os workflows científicos tendem a ser mais complexos e dinâmicos e, por serem projetados por cientistas, precisam adequar-se às necessidades de experimentos científicos com naturezas diversas. Por esse motivo é comum terem que lidar com ambientes heterogêneos, mesclando diferentes tecnologias, por exemplo, combinando execuções locais com execuções remotas de tarefas.

A solução desenvolvida no decorrer deste trabalho se propõe a lidar apenas com workflows científicos, portanto não serão analisados e discutidos workflows corporativos.

Os problemas com os quais a solução desenvolvida foi confrontada são aqueles focados em workflows científicos, como transferência e processamento de dados.

2.2 Computação Voluntária

A história da computação voluntária começa junto com a publicação do primeiro sistema que abertamente utilizava recursos de computadores voluntários, o *Great Internet Mersenne Prime Search* (GIMPS)[1] em 1996, que procurava por grandes números primos. O problema dos números primos é bastante leve (requer poucos recursos) em termos de tráfego, pois é enviado apenas um número e a resposta é afirmativa ou negativa, porém exige um processamento de média ou longa duração. A estratégia utilizada pelo GIMPS é bastante similar às utilizadas por Folding@home (LARSON et al., 2002) e ao SETI@home (ANDERSON et al., 2002) que são sistemas que conseguiram alcançar um grande poder computacional com a computação voluntária (ANDERSON, 2004). Os problemas com estas características (baixo tráfego de dados e processamento de média ou longa duração) se adequaram bem às características da computação voluntária.

A base fundamental da computação voluntária (BEBERG et al., 2009) é simples se comparada com suas questões em aberto, relacionadas à sua implantação e/ou aplicação (DETHIER et al., 2008; DUAN et al., 2012; DORNEMANN et al., 2012; DIAS et al., 2010). Antes de mais nada deve existir um projeto que aceite utilizar o processamento de outros computadores, chamados de voluntários (ou *workers*). Em geral esses computadores são pessoais e não identificados. O projeto deve estar bem dividido em partes menores, estas partes são chamadas de atividades (*tasks*, *jobs* ou *work units*), cada atividade contém uma função ou um programa, além dos dados de entrada necessários. O responsável então publica o projeto para que os voluntários o acessem (diz-se que o voluntários são "recrutados"). Os voluntários baixam uma atividade de cada vez e, em seguida, a processam e enviam os resultados de cada atividade de volta ao projeto.

A natureza da computação voluntária é extremamente volátil (DIAS et al., 2010). Os projetos, em sua maioria, estão alocados na internet, o que implica em problemas como instabilidade de conexões, velocidade de transmissão e atraso de envio. Esses problemas limitam a abrangência dos projetos, pois projetos com atividades muito longas ou com alto tráfego de dados podem ser pouco eficientes. Em geral os projetos possuem atividades

[1] http://www.mersenne.org

formadas de arquivos pequenos que gastam no máximo algumas horas de processamento. Além disso, os sistemas também são bastante dinâmicos, um voluntário pode entrar e sair do sistema quando bem entender, seja durante uma transmissão ou um processamento. Por não utilizar computadores identificados, alguns sistemas enviam a mesma atividade para vários voluntários, isso evita que algum comportamento malicioso distorça o resultado do projeto e evita os problemas relacionados a voluntários saindo do sistema durante a execução de atividades.

2.3 Desktop Grids

As estruturas e os processos de um *desktop grid* são bem parecidos com de um projeto de computação voluntária (SCIENCETALK, 2011). A principal diferença está na natureza volátil dos participantes na computação voluntária. Um desktop grid não trabalha com voluntários, mas o mesmo conceito de *worker* é utilizado. Os computadores participantes do grid são de conhecimento do responsável pelo projeto e estes são mantidos em um ambiente controlado. Um *worker* não sai do sistema sem o conhecimento do responsável do projeto e, por mais que ainda possam existir falhas, estas são mais raras.

2.4 Composição de Serviços

A composição de serviços tem o objetivo de combinar serviços para criar aplicações mais complexas. Pode-se dividir a composição de serviços em duas classes: orquestrada e coreografada.

A base da composição de serviços pode ser, em alguns pontos, comparada a computação voluntária. O programa interessado em um serviço faz uma requisição, enviando os parâmetros necessários, para que o serviço execute algum tipo de processamento e retorne uma saída. Este comportamento é similar à computação voluntária, com diferença de que um serviço não pode executar um processamento diferente daquele que foi previamente definido, enquanto que na computação voluntária tipicamente são enviados a requisição e o código que deve ser processado.

Na orquestração de serviços a resposta do serviço é retornada a quem fez a requisição. Já no caso de coreografia, os serviços se comunicam entre si e as saídas podem ser direcionadas para servirem de entradas para outros serviços. Este comportamento coreo-

grafado possibilitou melhorias significativas em termos de escalabilidade e transferência de dados (DUAN et al., 2012).

A aplicação de técnicas de computação P2P à computação voluntária parte de um princípio similar a coreografia de serviços, isto é, possibilitar a comunicação entre os diversos voluntários do sistema sem precisar passar por um coordenador (ou orquestrador) do projeto.

2.5 Escalonamento com ponto-a-ponto

Diversos trabalhos na literatura já chamam a computação voluntária de *P2P Computing* (entre outros nomes), isso se deve ao fato de na computação voluntária haver a transferência de ações/responsabilidades de uma entidade (responsável pelo projeto) para outras (voluntários). Porém, de um modo geral, a arquitetura da maioria dos sistemas que usam computação voluntária segue o esquema cliente-servidor.

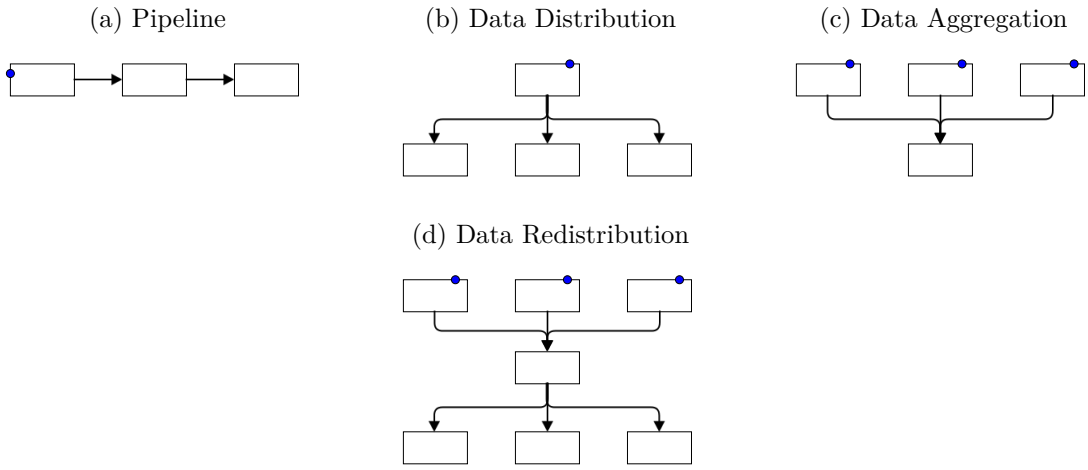
Os voluntários não se comunicam de forma descentralizada, isto é, para conseguir algum tipo de recurso (código ou dados) não é possível consegui-lo por meio de outro participante que não seja o servidor. Este mestrado atuou justamente nesta questão: permitir que os voluntários se comuniquem entre si com o intuito de diminuir a sobrecarga do servidor.

No Capítulo 3 são apresentados e discutidos outros trabalhos disponíveis na literatura onde os voluntários se comunicam entre si.

2.6 Benchmark

Durante a revisão na literatura não foram encontrados métodos amplamente utilizados para medir o desempenho computacional da execução de workflows científicos. Por isso foi utilizado um conjunto de workflows que são exemplos simplificados (*toy examples*), baseados em algumas estruturas básicas de workflows. Estas estruturas básicas foram definidas em (BHARATHI et al., 2008) e estão exibidas e nomeadas na figura 1. Os autores do referido artigo deixam claro que o número de atividades pode variar em cada uma destas estruturas. No decorrer deste trabalho as estruturas utilizadas para medir o desempenho seguem exatamente a mesma quantidade de atividades e estrutura utilizadas

Figura 1 – Estruturas básicas de workflows



Fonte: Caio Rafael do Nascimento Santiago, 2015

pelos autores e representadas na figura 1, variando apenas na quantidade de cópias das estruturas utilizadas nos testes (conforme será detalhado no capítulo 5).

Essas estruturas básicas representam a forma como as atividades, chamadas de *Jobs* por Bharathi et al. (2008), se relacionam com os dados. Na figura 1 as atividades são os retângulos e os dados são fornecidos pelas saídas das atividades ou indicados como já disponíveis pelos círculos azuis.

2.7 Infraestrutura

Este trabalho utiliza e expande a infraestrutura criada no decorrer do doutorado do orientador no Instituto de Computação da UNICAMP e estendido dentro do projeto Jovem Pesquisador em Centros Emergentes – FAPESP do orientador, intitulado "Gerenciamento e Composição Automática de Serviços Web Semânticos". Os próximos parágrafos apresentam brevemente o projeto Jovem Pesquisador.

A arquitetura do projeto foi originalmente prototipada para a especificação e anotação de workflows científicos em bioinformática. Os workflows são armazenados em bancos de dados e as ontologias são utilizadas para aumentar a semântica na composição automática de workflows. Além disso, foram inseridas funcionalidades ligadas a rastreabilidade de experimentos usando um modelo de proveniência de dados e serviços para permitir o armazenamento eficiente da proveniência dos experimentos (BARGA; DIGIAMPIETRI, 2008; DIGIAMPIETRI; PEREZ-ALCAZAR; MEDEIROS, 2007).

O projeto Jovem Pesquisador visou a estender a pesquisa realizada durante o doutorado do orientador, enfocando na combinação de diferentes técnicas de Inteligência Artificial (IA) para a composição de serviços; execução dos serviços compostos, estudo do melhor modelo ontológico a ser usado pela arquitetura para possibilitar o gerenciamento de informação semântica; uso de padrões de workflows; e desenvolvimento de técnicas para correção, em tempo de execução, de serviços defeituosos. Além disso, nesse projeto o conceito de atividade foi generalizado permitindo-se que não sejam utilizados apenas serviços web, mas também classes desenvolvidas na linguagem de programação Java e aplicativos locais. Esta generalização ampliou a disponibilidade das atividades para criação dos workflows permitindo que a infraestrutura proposta seja facilmente utilizada em diversos domínios (DIGIAMPIETRI et al., 2011; DIGIAMPIETRI et al., 2012; DIGIAMPIETRI et al., 2013; DIGIAMPIETRI et al., 2014b; ZUNIGA et al., 2014)

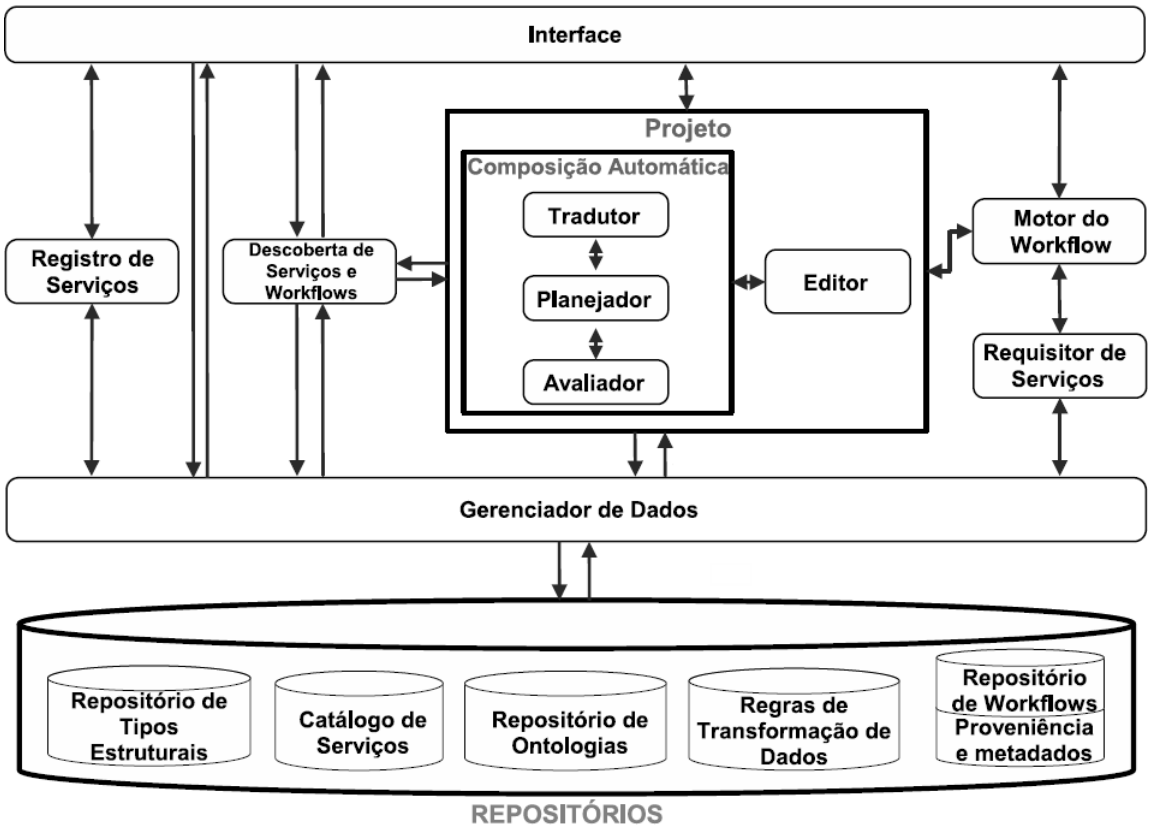
A figura 2 apresenta a arquitetura geral do projeto Jovem Pesquisador, que neste trabalho de mestrado foi expandida para aumentar a capacidade de modularização do "Motor do Workflow", tornando possível a utilização de diversos algoritmos de escalonamento e a comunicação com computadores voluntários.

2.8 Considerações Finais

Este capítulo apresentou os conceitos fundamentais que foram utilizados para a elaboração deste trabalho. Pelo fato do enfoque deste trabalho ser no processamento de workflows científicos, os problemas que enfocam alto tráfego de dados e longo processamento. Também foram apresentados alguns conceitos da computação voluntária, das questões que envolvem a técnicas de computação ponto-a-ponto, além de como a inspiração em coreografia de serviços poderá ser empregada na melhora do desempenho de sistemas de computação voluntária.

O capítulo seguiu com a descrição das estruturas básicas de workflows que serviram de base na avaliação do desempenho dos métodos de escalonamento. Por fim, o capítulo foi encerrado com a apresentação da infraestrutura que serviu como base para a criação da infraestrutura utilizada para a construção dos métodos de escalonamento.

Figura 2 – Arquitetura do projeto Jovem Pesquisador



Fonte: Luciano Antonio Digiampietri, 2007

3 REVISÃO SISTEMÁTICA

Este capítulo sumariza os principais aspectos da revisão sistemática realizada para identificar quais algoritmos de escalonamento podem ser utilizados para a computação voluntária a fim de tomar vantagem da computação ociosa. O texto completo da revisão sistemática está disponível na web[1].

3.1 Resumo da Revisão

A revisão sistemática foi realizada em novembro de 2013 visando a identificar quais algoritmos de escalonamento podem ser utilizados para a computação voluntária, quais as estruturas de dados utilizadas e técnicas de avaliação/validação empregadas.

Três bibliotecas digitais foram consultadas: ACM Digital Library[2], IEEE Xplore[3], ScienceDirect[4] e a busca originalmente resultou em 97 artigos. Porém, após a aplicação dos critérios de inclusão e exclusão apenas sete artigos foram selecionados. Os detalhes sobre as *strings* de busca utilizadas e critérios podem ser encontrados no relatório técnico produzido[1].

As subseções a seguir apresentam as principais características dos artigos selecionados e a conclusão deste capítulo identifica alguns pontos da revisão que foram considerados no desenvolvimento deste mestrado.

3.1.1 Estrutura de dados

Para entender melhor como os computadores estão organizados e como trabalham em cada um dos artigos as seguintes questões foram levantadas:

- Q1 - Como os nós se relacionam entre si?
- Q2 - Quais os diversos tipos de classes dos nós?
  - Q2.1 - Qual o papel de cada classe?
- Q3 - As atividades dever ser independentes?

[1] http://www.each.usp.br/digiampietri/relatorios/revisaosistematica_crns.pdf
[2] http://dl.acm.org
[3] http://ieeexplore.ieee.org
[4] http://sciencedirect.com

- Q4 - Existe algum meio que centralize parcialmente o escalonamento?
- Q5 - Existe algum tipo de *cache* de dados nos nós? E como é feito?
- Q6 - Existe algum tipo de *checkpoint* sobre os estados do processamento corrente?
  - Q6.1 - Como é feito o checkpoint?

Na tabela 1 estão as repostas a estas questões, extraídas dos artigos analisados.

Tabela 1 – Dados extraídos dos artigos referentes à estrutura de dados

ARTIGO	Q1	Q2	Q2.1	Q3	Q4	Q5	Q6	Q6.1
(CELAYA; ARRONATEGUI, 2010)	VBI-Tree (JAGADISH, 2006)	Routing node	Possui informações sobre a disponibilidade de recurso do Execution nodes	Não	Não	NA	Não	NA
		Execution node	Executa os jobs					
		Submission node	Criador das atividades e pelo escalonamento inicial					
(KWAN; JOGESH, 2010)	Super-peer	Super peer	Escalonam atividades para os worker	Sim	Sim	Não	NA	NA
		Worker	Processam atividades					
		Client	Criador das atividades e pelo escalonamento inicial					
Murata 2008(MURATA et al., 2008)	Vizinhança	BOINC	É a fonte inicial das atividades	Sim	Sim	NA	Não	NA
		Workers	Processam atividades e podem reescalonar atividade para outros workers					
(Wen Dou et al., 2003)	Vizinhança	Não possui classes	NA	Sim	Não	NA	Não	NA
(ZHAO; YANG; XU, 2009)	Vizinhança	Não possui classes	NA	Não	Não	NA	Não	NA
(RIUS et al., 2012)	Super-peer	Master	Criador das atividades e pelo escalonamento inicial	Sim	Sim	Não	NA	NA
		Manager	Escalonam atividades para os workers					
		Worker	Processam atividades					
(MASTROIANNI et al., 2009)	Super-peer	Super-peer	Age como *backbone* dos Super-peers, e junta o workers aos jobs	Sim	Sim	Sim	Cache sobre jobs e dados, estes são armazenados no Data cachers para serem executados múltiplas vezes	NA
		Data cacher	Disponibiliza os jobs e os dados para os Workers					
		Data source	Recebe os dados de fontes externas e transmite para os Data cachers e para o Workers					
		Job manager	Disponibiliza os job para os Super-peers e Data cachers					
		Worker	Processam atividades					

Fonte: Caio Rafael do Nascimento Santiago, 2015

Pode-se notar que existe um equilíbrio entre os artigos que utilizam estratégias baseadas em vizinhança e *super-peer*. A vizinhança nestes casos resulta em uma modelagem de grafos enquanto que os super-peers se aproximam mais de uma modelagem baseada em

árvores, assim como a estratégia baseada na VBI-Tree. Uma exceção a isso é o caso (RIUS et al., 2012) em que existem tantos elementos envolvidos no sistema que a modelagem se aproxima tanto de árvores quanto de grafos. Devido a modelagem utilizada nestes artigos, os computadores são tipicamente chamados de nós ou *peers*. A forma como os computadores se relacionam nas propostas está intimamente ligada à existência de algum meio que centralize parcialmente o escalonamento, pois os artigos baseados em super-peers estão de alguma forma centralizando o escalonamento, além destes, no trabalho de Murata et al. (2008) é apresentada uma proposta dependente do BOINC (ANDERSON, 2004) pois todas as atividades partem inicialmente do BOINC.

Quanto às classes, apenas dois artigos não as utilizam, (Wen Dou et al., 2003) e (ZHAO; YANG; XU, 2009). Nestes trabalhos os computadores também se relacionam por meio de vizinhanças, as estratégias de escalonamento são mais simples e baseadas em decisões locais. Entre os vários nomes que cada um dos artigos escolheu, um tornou-se recorrente, são chamados de *worker* os computadores responsáveis por processar as atividades (também chamadas de *Jobs* pela maioria dos artigos). Algumas destas classes não são computadores voluntários (em geral as classes que não são workers), mas isso pode ser um fator complicador, pois exigem computadores dedicados e que apresentam um fator maior de confiança. Quanto maior o número de classes maior é o esforço inicial do responsável pelo sistema, por exemplo, a solução proposta por Mastroianni et al. (2009) tem uma configuração inicial com quatro computadores.

A maioria dos artigos trabalha apenas com atividades independentes, i.e., não possuem uma estratégia diferente para atividades dependentes. No trabalho de Zhao, Yang e Xu (2009) uma limitação é que as atividades devem ser todas dependentes, este algoritmo trabalha apenas com problemas divisíveis (problemas de divisão e conquista). No trabalho de Celaya e Arronategui (2010) as atividades podem ser independentes mas isso pode não ser tão eficiente uma vez que o algoritmo de escalonamento está melhor preparado para trabalhar com atividades dependentes. A maioria dos artigos trabalha com atividades do tipo *Bag-of-Task*, i.e., um conjunto de atividades que não se comunicam e são independentes.

O cache de dados e o checkpoint do processamento são questões importantes, pois algumas atividades podem ter um custo muito grande tanto de transferência de dados quanto de processamento. Nenhum artigo tratou de questões de checkpoints como forma de minimizar os danos da ocorrência de falhas, i.e., quando um computador faz *backups*

periódicos do estado das variáveis da atividade que esteja processando. Também não têm formas preventivas de lidar com falhas como, por exemplo, no trabalho de Zhou e Lo (2006). Apenas um artigo aplica cache (RIUS et al., 2012), o cache fica nos *Data cachers* que funcionam com *Job managers* e *Data sources*, mas aparentemente o sistema não está preparado para aceitar computadores voluntários para essa tarefa, portanto em nenhum dos sistemas os voluntários funcionam como cachers do sistema.

3.1.2 Algoritmo de escalonamento

Os algoritmos de escalonamento não puderam ser descritos em perguntas simples e, portanto, serão descritos de forma extensa.

Os algoritmos propostos por Wen Dou et al. (2003) e Zhao, Yang e Xu (2009) são relativamente parecidos e serão descritos juntos. Nestes casos não existem classes e os computadores tanto processam quanto escalonam atividades para seus vizinhos. No trabalho de Wen Dou et al. (2003) as atividades já estão todas distribuídas no sistema e um computador, quando sobrecarregado, escalona uma atividade para o vizinho menos sobrecarregado. Já no trabalho Zhao, Yang e Xu (2009) o computador recebe uma atividade e a divide, em seguida transmite os filhos da atividade para seus vizinhos. Este algoritmo trabalha com uma classe muito limitada de algoritmos e não fica claro se esta solução é eficiente em todos os casos, por exemplo, se considerarmos o *merge sort* no qual as etapas de divisão, conquista e combinação são rápidas e a transferência de dados pode ser custosa esse algoritmo não dá garantias de eficiência. Ambos os algoritmos são simples e executam transferências baseadas em decisões locais e, por isso, também podem acarretar em problemas de mínimos locais.

Os algoritmos apresentados por Kwan e Jogesh (2010) e Rius et al. (2012) também possuem algumas características em comum. Ambos possuem três tipos de classes: uma que é a fonte das atividades, uma que processa as atividades e *super-peers* que agem entre a primeira e a segunda classe. O escalonamento é feito em duas partes, na primeira, a fonte das atividades escalona a atividade para o super-peer, em seguida o super-peer escalona a atividade para o computador que a processará. No algoritmo de Kwan e Jogesh (2010), a primeira etapa é probabilística, onde o super-peer com menor sobrecarga tem maior probabilidade de receber uma atividade. Os super-peers se comunicam entre si trocando informação sobre as suas próprias cargas, a segunda o super-peer escolhe o melhor

computador dentro de um *deadline*. Já na primeira etapa do algoritmo de Rius et al. (2012) é utiliza uma fórmula que combina a reputação, o centro de massa e a capacidade computacional do vizinhos para escolher o super-peer, e na segunda etapa o super-peer escalona a atividade para o computador com o menor custo.

O algoritmo de escalonamento apresentado por Murata et al. (2008) é uma extensão do BOINC (ANDERSON, 2004), que já é amplamente conhecido e utilizado. A diferença é que os computadores que processam as atividades se organizam em subgrupos e dentro de cada subgrupo todos se comunicam entre si. O BOINC continua fornecendo as atividades, mas quando um computador está sobrecarregado ele re-escalona uma atividade para um membro de seu subgrupo que esteja mais livre. Isso diminui a taxa de transferência do BOINC e consequentemente aumenta a escalabilidade do sistema.

A estratégia adotada por Mastroianni et al. (2009) é relativamente simples. Os computadores que processam as atividades (*workers*) as pedem para os *Super-peers* e estes repassam o pedido para os *Data cachers* que completam o envio. Os *Data cachers* por sua vez recebem as atividades do *Job managers*. Os *workers* também devem pedir os dados, o processo é o mesmo, mas em vez dos *Data cachers* receberem os dados do *Job manager* eles enviam o pedido para o *Data source* que o repassa para o *worker*. As respostas dos *workers* são enviadas para o *Job manager*.

Por fim, a abordagem proposta por Celaya e Arronategui (2010) utiliza uma estratégia de árvore que possui três classes: a raiz é a fonte das atividades (*submission node*), os galhos são os *routing nodes*, os galhos e as folhas processam as atividades (*execution nodes*). O problema é descrito como um DAG (*Directed acyclic graph*), o *submission node* separa o caminho mais longo e o envia para os primeiros *routing nodes*. Cada *routing node* possui uma sumarização dos dados de disponibilidade nos *execution nodes* que estão abaixo dele, com base nesta informação ele escalona a lista de atividades dentro das janelas de disponibilidade, a qual ele conhece de forma gulosa (similar ao problema da agenda), isto é, repassando as atividades para baixo na topologia de árvore. Os *routing nodes* fazem isso repetidamente até que as atividades cheguem às folhas.

3.1.3 Avaliação do escalonamento

Assim como na análise feita sobre a estrutura de dados, para analisar a avaliação do escalonamento perguntas foram preparadas, são elas:

- Q1 - O método de avaliação do escalonamento foi uma simulação?
- Q2 - Qual é a configuração e quantos computadores foram envolvidos nos testes?
- Q3 - Qual a conexão entre os computadores?
- Q4 - A avaliação tratou simulação de falhas?
- Q5 - Qual foi o problema abordado na avaliação?
- Q6 - Quais métricas foram utilizadas para medir o desempenho do escalonamento?

Na tabela 2 estão respostas a estas questões extraídas dos artigos analisados.

Tabela 2 – Dados extraídos dos artigos referentes à avaliação do escalonamento

Artigo	Q1	Q2	Q3	Q4	Q5	Q6
(CELAYA; ARRONATEGUI, 2010)	Sim	100 Peers	1Mbps 50ms (latencia)	Não	Fork-Join e Laplace equation	Workflow relative priority
(KWAN; JOGESH, 2010)	Sim	4.000 Peers	50 segundos para qualquer transferência	Não	Número fixo de instruções (25.000 e 75.000)	Task Turnaround, Packet overhead per Task
(MURATA et al., 2008)	Não	Server Core2Duo InTrigger (2008)	100Mbps (servidor)	Não	0 e 100 segundos de processamento	Server-CPU usage, Network load, The turnaround time of a Work Unit
(Wen Dou et al., 2003)	Sim	4.000 peers	NA	Sim	Não abordou. Apenas verificou o deslocamento das atividades	NA
(ZHAO; YANG; XU, 2009)	Não	3 computadores	Não especificado	Não	N-Rainhas	Tempo de execução combinada e eficiência
(RIUS et al., 2012)	Sim	1.000.000 Wokers 54 Manager	Não especificado	Não	Não especificado	Balance
(MASTROIANNI et al., 2009)	Sim	1.000 Workers 100 Super-peers	1Mbps entre Super-peers 10Mbps entre Super-peers e workers 100 ms (latência) entre Super peers 10 ms (latencia) entre Super-peers e workers	Sim	Ondas Gravitacionais (GridOneD)	Tempo de execução

Fonte: Caio Rafael do Nascimento Santiago, 2015

Pode-se notar que a maioria dos testes foram simulações e isso é plausível considerando-se o montante de computadores em alguns testes seria necessária uma quantidade muito grande de recursos para envolver 10.000.000 de computadores reais (CELAYA; ARRONATEGUI, 2010).

Apenas dois artigos não fizeram simulações. No trabalho de Murata et al. (2008) foram utilizados um computador comum como fonte das atividades (um computador

Intel Core2Duo), já os outros participantes são os computadores que compunham o InTrigger[5] em 2008. O InTrigger é um grid formado por computadores pessoais localizado no japão, o grid está em constante evolução, mas na época deste experimento possuía 222 computadores, mas as configurações de rede não foram especificadas nas condições da época. O outro trabalho (ZHAO; YANG; XU, 2009) a não realizar uma simulação utilizou três computadores não especificados e também não especificou as configurações de rede utilizadas.

Os artigos que fizeram simulações são abrangentes nos elementos envolvidos em seus experimentos. O trabalho de Rius et al. (2012) foi o que mais utilizou computadores na avaliação do escalonamento (1.000.000 como workers e mais 54 computadores como managers). As configurações de rede não foram informadas.

Três trabalhos utilizam centenas ou milhares de computadores (KWAN; JOGESH, 2010; Wen Dou et al., 2003; MASTROIANNI et al., 2009), sendo que no trabalho de Mastroianni et al. (2009) também foram simulados 100 *super-peers*. Já as configurações de rede são bem distintas. Kwan e Jogesh (2010) fixaram todas as transferências em um tempo fixo de 50ms. Wen Dou et al. (2003) não informaram, e Mastroianni et al. (2009) utilizam configurações separadas entre os super-peers e dos super-peers para o workers, respectivamente são: 1Mbps com latência de 100ms e 10Mbps com latência de 10ms.

Por fim, o artigo que simulou a menor quantidade de computadores foi o de Celaya e Arronategui (2010), no qual foram simulados 100 computadores conectados em por uma banda de 1Mbps com latência de 50ms.

Quanto a simulação de falhas, apenas dois trabalhos (MASTROIANNI et al., 2009; Wen Dou et al., 2003) realmente fazem simulações de falha, nos outros até são apresentados mecanismos de tratamento de falhas, mas os autores não fizeram experimentos demonstrando sua eficiência.

Os problemas abordados durante a avaliação dos escalonadores foram dos mais variados. Wen Dou et al. (2003) não utilizaram um problema específico, seu objetivo era prova que o caminho mais curto de um computador a outro era de no máximo sete intermediários, isso em si não é uma forma de se comprovar a eficiência do algoritmo. Kwan e Jogesh (2010) utilizaram várias atividades de tamanho fixo de 25.000 e 75.000 instruções e avaliou com o *Task Turnaround* e o *Packet Overhead per Task*, por utilizar o deadline na composição do algoritmo o problema abordado é muito propício, pois poucos problemas

[5] http://www.intrigger.jp/

reais podem ter seu deadline facilmente calculado. Murata et al. (2008) utilizaram uma abordagem similar, mas fixando o tempo de processamento de cada atividade a uma constante, e avaliou com o *Server-CPU usage* e o *turnaround time of a Work Unit*. Rius et al. (2012) não especificou um problema e avaliou com *Scheduling Overhead* e o *Load Balance*. Os artigos (MASTROIANNI et al., 2009), (CELAYA; ARRONATEGUI, 2010) e (ZHAO; YANG; XU, 2009) são mais realísticos. Mastroianni et al. (2009) utilizaram *Ondas Gravitacionais* (GridOneD) avaliando o tempo de execução. Celaya e Arronategui (2010) utilizaram *Fork-Join* e *Laplace equations* e avalia com *workflow relative priority*. Zhao, Yang e Xu (2009) utilizaram o problema das N-Rainhas com o tempo de execução combinado dos três computadores e a eficiência.

3.2 Comparação com a Solução Desenvolvida no Mestrado

A revisão da literatura inspirou a solução proposta e desenvolvida neste mestrado tanto fornecendo algumas bases para a construção da solução como apresentando alguns problemas em aberto que, neste projeto, tentamos solucionar. Os próximos parágrafos apresentam um breve paralelo entre os trabalhos correlatos e a solução proposta. No próximo capítulo (Capítulo 4) será detalha a solução proposta.

A solução proposta e desenvolvida nesta dissertação utilizou algumas estratégias diferentes daquelas observadas na revisão. A principal diferença foi a construção de uma estratégia diferenciada para o compartilhamento/reutilização de parâmetros por meio da utilização de conceitos de computação P2P entre os computadores voluntários. Em situações em que os parâmetros são reutilizados a solução visa a diminuir a banda de *upload* do servidor e uma redução no tempo total do sistema. Isso ocorre porque a solução considera os parâmetros como estruturas independentes das atividades que os utilizam e ao fato dos voluntários guardarem e compartilharem os parâmetros de forma proativa enquanto estão processando outras atividades.

Celaya e Arronategui (2010) apresentaram uma estratégia voltada para o tratamento de atividades dependentes, mas a solução desenvolvida neste mestrado não diferencia atividades dependentes ou independentes, e, portanto, não se prejudica por trabalhar com um determinado tipo ou outro de atividades. Mesmo que a reutilização de parâmetros se mostre eficiente para a execução de atividades dependentes (pois a saída produzida é inevitavelmente reutilizada) a solução proposta neste mestrado não está limitada a elas.

Assim como nos trabalhos de Wen Dou et al. (2003) e Zhao, Yang e Xu (2009) a solução utiliza uma organização baseada em vizinhança e trabalha com decisões locais. Porém, diferentemente do trabalho de Zhao, Yang e Xu (2009) os dados transmitidos não se limitam aos últimos dados obtidos (além do sistema conseguir trabalhar livremente com atividades independentes). No trabalho de Wen Dou et al. (2003) as atividades são espalhadas pelos voluntários logo no início da execução (não sendo possível a execução de atividades dependentes) e são repassadas ao vizinhos conforme vão sendo processadas, mas não está claro se essas decisões de repasse são realizadas no momento em que o voluntário está processando a atividade.

Provavelmente o trabalho de Murata et al. (2008) seja o mais próximo da solução desenvolvida. Os voluntários baixam os dados do servidor e escalonam as atividades conforme a necessidade para seus vizinhos, contudo as atividades estão associadas aos parâmetros (nesse trabalho não é possível obter separadamente a atividade e os parâmetros). Não existe uma estratégia diferenciada para trabalhar com reutilização de parâmetros e nem para trabalhar com atividades dependentes.

3.3 Considerações Finais

Os artigos estudados apresentaram pouca diversidade de estrutura de dados, coisa que não acontece nos métodos de escalonamento e nos métodos de avaliação do desempenho dos escalonadores. Os métodos de escalonamento possuem desde soluções simples até as mais elaboradas e inteligentes, mas é difícil falar sobre a qualidade dos resultados destas soluções por causa da diversidade de métodos de avaliação. Por causa dos métodos de avaliação não foi possível comparar os resultados dos artigos vistos entre si e entre métodos que não utilizam técnicas de P2P. Os métodos de avaliação não seguiram nenhum padrão ou *benchmark*, o que poderia ser de grande ajuda para permitir uma comparação mais fidedigna entre as soluções.

Pôde-se perceber que existem poucos artigos sobre esse tópico, mostrando que é ainda um tópico incipiente e, portanto, um campo aberto para pesquisas. Talvez um motivo para explicar isso é que faz poucos anos que a internet chegou com velocidade relevante para usuários comuns a ponto de que estes pudessem baixar e transmitir dados desse tipo de sistema.

4 SOLUÇÃO DESENVOLVIDA

Neste capítulo é apresentada a solução desenvolvida, cujo cerne corresponde a um método de escalonamento para computação voluntária que utiliza técnicas de computação P2P.

Adicionalmente, são detalhadas as modificações realizadas sobre a infraestrutura desenvolvida no projeto Jovem Pesquisador em Centros Emergentes – FAPESP do orientador. A maioria das extensões realizadas teve por objetivo tornar a infraestrutura mais modular e flexível, capaz de lidar com dados distribuídos.

Neste capítulo também é explicada a forma com a qual os computadores voluntários se comunicam. E, por fim, o algoritmo de escalonamento desenvolvido é apresentado e explicado passo a passo. Para a explicação deste algoritmo, inicialmente são apresentadas as classes envolvidas no processo e, em seguida, a sequência de atividades e/ou trocas de mensagens que ocorrem entre voluntários ou entre um voluntário e o servidor de forma a se executar um workflow.

Os códigos fonte desenvolvidos neste mestrado, bem como dados de teste, encontram-se disponíveis na web[1].

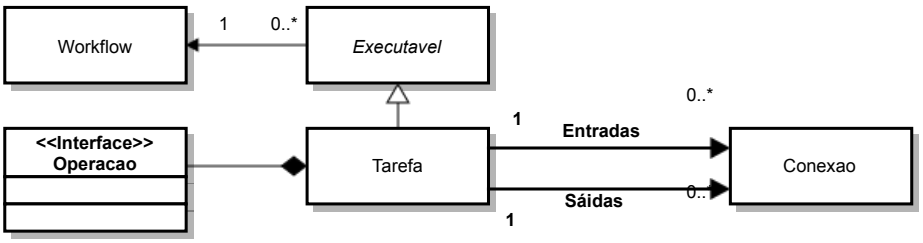
4.1 Extensão do Modelo de Representação de Workflows

Conforme apresentado na seção 2.7, este projeto estendeu um SGW existente. Na estrutura básica de representação do workflow desse SGW (Figura 3) era necessário que houvesse uma instância da classe *Workflow* se comunicando com uma instância da *FilaDeExecucao* para que esta então inicie o processamento. A classe Workflow detinha todo o conhecimento sobre as atividades (que são do tipo *Executavel*) e sobre o fluxo de dados entre cada uma delas. Apesar de esta estrutura ser bastante conveniente para a execução orquestrada de workflows simples, há algumas características que a tornam ineficiente na execução de workflows mais complexos e execução distribuída das atividades. Por exemplo, ela impossibilitava a criação de estruturas de controle de fluxo com laços ou condicionais de forma eficiente, além de impossibilitar que partes de um workflow executem sem o conhecimento completo do workflow.

[1] http://www.each.usp.br/digiampietri/codigos/codigos_crns.7z

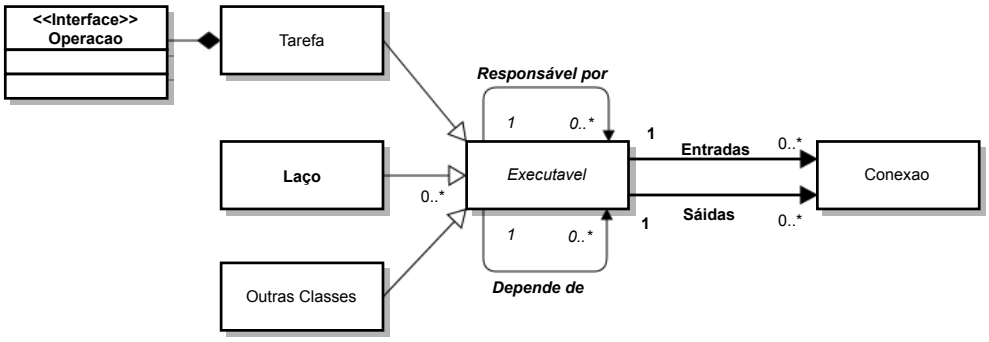
Para lidar com as necessidade deste trabalho foram realizadas algumas modificações na estrutura básica de representação do workflow, como pode ser observado na figura 4. A classe *Executavel* recebeu as informações sobre o controle de fluxo que estavam armazenar em *Workflow* e cada atividade recebeu as informações referentes a si própria. As atividades agora sabem quais outras atividades precisam ser concluídas antes de começarem a executar (*Depende de*) e quais atividades devem ser avisadas quando forem concluídas (*Responsável por*). Anteriormente, apenas a classe *Tarefa*, que é um tipo particular de *Executavel*, possuía informações de entradas e saídas, mas atualmente todo *Executavel* possui essas informações, isso facilita a criação de outras classes que herdem *Executavel*. Com esta estrutura foi mais fácil criar outras classes para controle de fluxo (*Loop*, *If* e *Or*). O *Loop*, por exemplo, necessita criar novas instâncias de *Executaveis* em tempo de execução, o que não seria possível na estrutura anterior.

Figura 3 – Diagrama de classes da estrutura de representação do workflow



Fonte: Caio Rafael do Nascimento Santiago, 2015

Figura 4 – Diagrama de classes da estrutura modificada de representação do workflow



Fonte: Caio Rafael do Nascimento Santiago, 2015

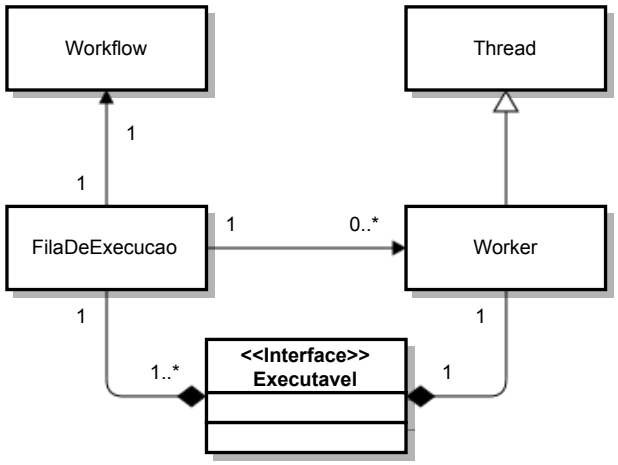
A exclusão da classe *Workflow* também causou impactos no executor de workflows, como pode ser observado pelas diferenças entre as figuras 5 e 6. A classe *FilaDeExecucao*

não precisa mais das informações contidas em *Workflow*, basta saber quais atividades devem ser executadas logo no início do processo. Esta classe se responsabiliza apenas por executar os *Executáveis* marcados como prontos e marcar como prontos os próximos *Executáveis* que são ativados quando algum acaba, deixando assim a execução mais descentralizada e dinâmica.

A simplificação de todas estas estruturas e do algoritmo de escalonamento das atividades permitiu a abstração da classe *FilaDeExecucao*, facilitando assim o desenvolvimento de novos algoritmos de escalonamento e a escolha dentre os que estão disponíveis. Os processos de lidar com as atividades concluídas e habilitar as próximas atividades que podem ser executadas ficaram a cargo da classe abstrata, já o processo de escolha e execução das atividades ficou a cargo das classes filhas.

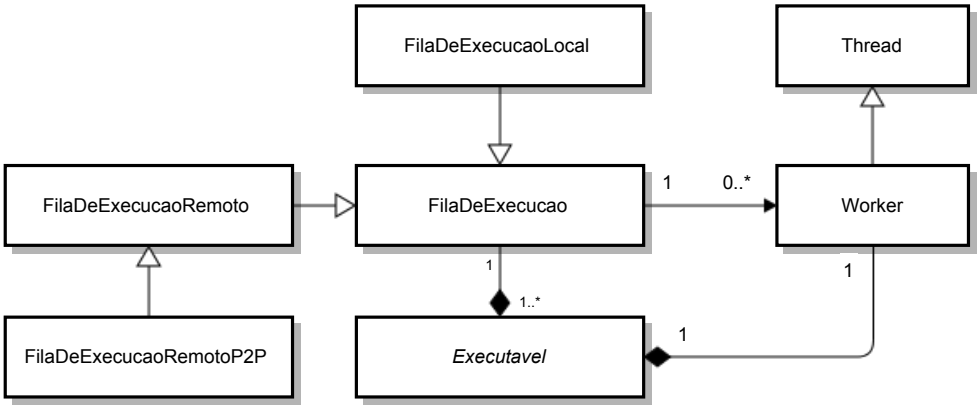
Foram implementados três algoritmos de escalonamento diferentes: um que utiliza apenas a execução local, tendo como estratégia o caminho mais longo; um baseado em sistemas de computação voluntária; e, por fim, o algoritmo que utiliza computação voluntária combinada com técnicas de computação P2P.

Figura 5 – Diagrama de classes da estrutura do executor Workflow



Fonte: Caio Rafael do Nascimento Santiago, 2015

Figura 6 – Diagrama de classes da estrutura modificada do executor Workflow

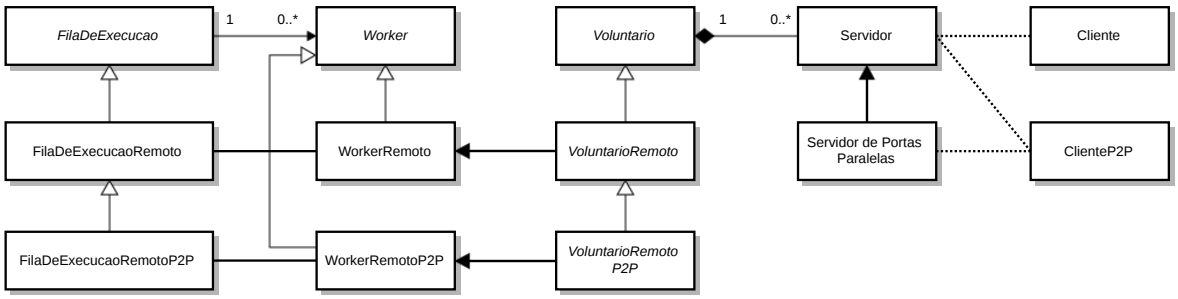


Fonte: Caio Rafael do Nascimento Santiago, 2015

4.2 Comunicação entre Computadores

O algoritmo de escalonamento que combina computação voluntária com P2P exige que os computadores voluntários sejam capazes de se comunicar com outros voluntários. Para esta comunicação, foi desenvolvida a estrutura definida pelo diagrama da figura 7.

Figura 7 – Diagrama de classes da estrutura de comunicação entre o servidor e os voluntários



Fonte: Caio Rafael do Nascimento Santiago, 2015

Foram criadas duas aplicações separadas para desempenhar o papel dos clientes no sistema. A classe *Cliente* desempenha o papel do cliente na computação voluntária tradicional, enquanto que a classe *ClienteP2P* desempenha este papel na computação voluntária aplicando as técnicas de computação P2P. Ambas as classes se comunicam com a classe *Servidor*, que é a responsável por "recrutar" os voluntários e disponibilizá-los para a *FilaDeExecucao* (onde todo o processo de escalonamento acontece).

A execução das atividades depende de que os voluntários recebam no mínimo três elementos que precisam ser serializados para serem transmitidos. O primeiro elemento é a atividade em si, como a estrutura toda é baseada em Java, a atividades é, em geral, uma função contida em uma classe. O segundo elemento são as classes associadas à classe principal da atividade, pois muitas vezes a execução de uma classe faz referências a outras classes e, portanto, estas classes precisam estar carregadas na memória antes de serem chamadas. Por último, os voluntários precisam receber as entradas das atividades. As classes foram serializadas em um processo de duas etapas: na primeira a definição da classe é adquirida em tempo de execução utilizando técnicas de reflexão; e na segunda, as instâncias dos objetos foram serializados utilizando o *XStream*[2], uma biblioteca especializada para a serialização de objetos em Java. Quanto à serialização das entradas, foi necessária a criação de uma interface a ser utilizada pelo projetista do workflow para indicar o método de serialização dos dados, por causa da diversa gama de estruturas possíveis.

Os dois tipos de clientes se comunicam com o servidor por meio de mensagens, mas a diferença entre os dois é basicamente comportamental. A classe *Cliente* é passiva, pois possui um conhecimento restrito a uma atividade por vez e se comunica com o servidor apenas respondendo as mensagens. Já a classe *ClienteP2P* é ativa, o servidor que assume o papel de responder as mensagens do cliente e, além disso, também possui um conhecimento mais amplo do workflow e se comunica com outros voluntários para enviar e receber as entradas das atividades.

A classe *Cliente* baixa as entradas do servidor de forma continua. Já a classe *ClienteP2P* quebra as entradas em "pedaços" de tamanhos iguais, assim cada pedaço de uma entrada pode ser enviado por voluntários diferentes.

Para realizar a comunicação entre os voluntários foi criado um servidor auxiliar (o Servidor de portas paralelas), esse mecanismo ajuda os voluntários a abrirem portas, isto é, o voluntário envia um pacote de *datagrama* e o servidor informa os dados referentes à porta aberta pelo *socket* que enviou o datagrama. Ciente das portas abertas o voluntário já está apto a se comunicar com os outros utilizando estas portas. Por intermédio do servidor os clientes sinalizam o interesse em uma conexão para em seguida receberem os dados referentes a outro voluntário com uma porta aberta. Com esses dados, o voluntário inicia uma conexão utilizando o TseudoTCP. O *Jitsi*[3] é uma aplicação de comunicação por

[2] http://xstream.codehaus.org/
[3] http://jitsi.org/

áudio e vídeo feita em Java, e sua comunidade mantém um *framework* chamado *Ice4j*[4] que entre outras coisas contém o TseudoTCP. O TseudoTCP trabalha sobre o protocolo UDP e age na camada de aplicação para simular o protocolo TCP, essa estratégia permite estabelecer uma conexão utilizando apenas sockets UDP e, portanto, é possível estabelecer uma conexão sem a existência de uma porta fixa.

4.3 Computação Voluntária aplicando técnicas de Ponto-a-Ponto

O Algoritmo 1 mostra, em alto nível, o funcionamento da classe ClienteP2P, que é a responsável pela execução das atividades nos computadores voluntários aplicando P2P. A primeira diferença significativa é que o voluntário possui informações mais gerais sobre o workflow. Os objetos esquemáticos são uma simplificação das atividades, eles possuem apenas as identificações e as relações das atividades e servem para nortear o escalonamento das atividades. O cliente inicia o algoritmo requisitando uma lista de objetos esquemáticos, pois é a partir dessa lista que serão tomadas as decisões de escalonamento. Após se escolher a atividade (na Função *Escolher melhor atividade*) o cliente baixa os parâmetros e demais dados faltantes, inicia uma função em Segundo Plano e começa o processamento da atividade. Quando a atividade termina é enviado ao servidor os dados resultantes, e esses dados são marcados como *não confirmados*. Esta marcação ocorre porque, em uma situação em que o servidor exige redundância sobre os resultados das atividades, todas as atividades precisam terminar para o servidor saber se os resultados são válidos e, portanto, possam ser reutilizados pelo voluntário que o gerou.

Os objetos esquemáticos são postos em uma fila de prioridade onde são organizados conforme uma lista de características, são elas:

- As atividades são posicionadas depois das atividades das quais dependem;
- As atividades que têm mais parâmetros não disponíveis vêm depois;
- As atividades que têm mais parâmetros disponíveis vêm antes;
- As atividades com menos itens faltando para serem baixados vêm antes;
- As atividades com menos itens faltando para serem baixados ou baixando vêm antes;
- As atividades que têm mais parâmetros não confiáveis baixados vêm antes;

[4] https://code.google.com/p/ice4j/

Algorithm 1 Algoritmo de Computação Voluntária aplicando técnicas de P2P – Funcionamento do algoritmo no voluntário

```
1: Conectar ao Servidor
2: Requisitar identificação
3: Abrir Portas
4: Requisitar objetos esquemáticos
5: Fila f = Criar fila de prioridade com os objetos esquemáticos
6: while f não está vazio do
7:     Objeto obj = Escolher melhor atividade(f)
8:     for all entradas ent de obj do
9:         if ent ainda não foi baixado then
10:            baixar ent
11:    Informar o servidor que iniciará a execução do obj
12:    if obj está disponível para executar then
13:        Baixar classes associadas a classe principal
14:        Baixar classe principal
15:        Atividade a = Baixar atividade
16:        Ativar Thread de Segundo Plano
17:        Executar a
18:        Parar Thread de Segundo Plano
19:        Enviar Resultado
20:        Marcar Resultado como Parâmetro não confirmado
```

Fonte: Caio Rafael do Nascimento Santiago, 2015

A partir desta relação de ordem entre as atividades o algoritmo sempre procura a melhor atividade para executar (Função *Escolher melhor atividade* presente no Algoritmo 2). Enquanto não existir nenhuma atividade disponível o algoritmo continua procurando parâmetros (entradas) para baixar. Se uma atividade se torna disponível para ser processada, o voluntário já está potencialmente mais preparado (por ter baixado os parâmetros com antecedência), uma estratégia mais proativa. Os voluntários procuram baixar os parâmetros, sempre que possível, dos seus vizinhos (caso eles os possuam). Se os parâmetros fossem baixados apenas nas vésperas de se executar uma atividade isso poderia significar numa perda de desempenho considerável, uma vez que a comunicação é potencialmente mais lenta, pois funciona de forma fragmentada, os objetos são baixados em pequenas partes por vez e existe uma maior concorrência pelos parâmetros.

Esta função também verifica se os parâmetros marcados como não confiáveis foram validados pelo servidor e se os *downloads* foram concluídos.

Depois de escolhida a atividade, resta baixar as informações a serem processadas. No momento em que a atividade começa a ser processada também se inicia (em para-

Algorithm 2 Escolha da melhor atividade para se processar

```
1: procedure ESCOLHER MELHOR ATIVIDADE(f)
2:     Lista l
3:     while true do
4:         for all Parâmetros não confiáveis do
5:             if Verificar com o servidor se o parâmetro já é confiável then
6:                 if Check sum do parâmetro é igual ao check sum do servidor then
7:                     Marcar parâmetro como confiável
8:                 else
9:                     Deletar parâmetro
10:        for all Donwloads do
11:            if download concluiu then
12:                Marcar parâmetro como confiável
13:        while f não está vazia & número de Downloads < limite de Downloads do
14:            Objeto obj = Retirar objeto de f
15:            if obj não foi concluído then
16:                Adiciona obj em l
17:                if obj possui algum parâmetro não disponível then
18:                    Verificar parâmetros de obj com o servidor
19:                disponível = Todos os parâmetros de obj estão disponíveis
20:                pronto = obj pronto para executar
21:                excesso = não há excesso de concorrência
22:                if disponível & pronto & excesso then
23:                    Adiciona l em f
24:                    return obj
25:                while (!pronto | excesso) & Downloads < limite de Downloads do
26:                    Baixar uma entrada disponível de obj
27:        Adiciona l em f
28:        Limpar l
```

Fonte: Caio Rafael do Nascimento Santiago, 2015

lelo) um processamento similar à função *Escolher melhor atividade*. Esse processamento (chamado no algoritmo como *Segundo Plano*) começa a preparar futuras atividades para serem processadas assim que a atividade atual terminar, a única diferença é que esse processamento não inicia a execução de uma atividade.

4.4 Considerações Finais

Neste capítulo foi apresentada a solução desenvolvida neste trabalho, um sistema de computação voluntária que aplica técnicas de computação P2P. Os voluntários se comunicam entre si a fim de trocarem dados dos parâmetros enquanto estão ocupados

processando as atividades. O objetivo disso é utilizar a banda de transmissão de dados dos voluntários para diminuir uma potencial sobrecarga de requisições ao servidor.

Para alcançar o objetivo deste projeto, foi necessário utilizar um SGW, descrito na seção 2.7, e modificá-lo para facilitar a modularização dos componentes, permitindo assim ao SGW trabalhar com diferentes métodos de escalonamento.

5 AVALIAÇÃO DA SOLUÇÃO E RESULTADOS

Neste Capítulo estão descritos os mecanismos utilizados para avaliar o desempenho da solução desenvolvida. Foram definidos os recursos de *hardware* e *software* utilizados para a execução de dois conjuntos de testes. O primeiro conjunto utiliza o *workflow* de um problema real de análise e rede sociais. Já o segundo o segundo utiliza exemplos simplificados de estruturas básicas de workflows que formam descritas na seção 2.6. Em seguida são descritos e discutidos os resultados obtidos.

5.1 Avaliação da Solução

A seguir são apresentadas as configurações (de infraestrutura, software e estudo de caso) utilizadas nos testes realizados.

Foram utilizados ao todo quatro computadores (*notebooks*), com as configurações definidas na tabela 3. Todos os computadores foram conectados em um roteador *Wireless* e tiveram suas larguras de banda limitadas via *software*. Para a execução do sistema foi necessária a instalação de Java® em todos os computadores (em conjunto com algumas bibliotecas). Adicionalmente foi necessária a instalação de um programa para realizar a limitação de banda via software, para isto foi utilizado o *trickle*[1], um programa para Linux que limita a banda de um determinado processo.

Tabela 3 – Configurações do computadores utilizados

#	Processador	# Threads	Clock	Memória	S.O.
1	Intel®Core i5-3230M	4	3.20 GHz	8 GB	Ubuntu 14.04
2	Intel®Core i3-350M	4	2.26 GHz	4 GB	Ubuntu 14.04
3	Intel®Pentium T3400	2	2.26 GHz	3 GB	Ubuntu 12.04
4	Intel®Core 2 Duo T5750	2	2.00 GHz	2 GB	Ubuntu 12.04

Fonte: Caio Rafael do Nascimento Santiago, 2015

As velocidades de conexão foram limitadas com o intuito de simular as condições reais de conexões com a internet, por isso foram utilizadas como base as velocidades médias das conexões brasileiras (2,4 Mbps de *download*)(AKAMAI, 2013), infelizmente não foi

[1] https://wiki.archlinux.org/index.php/Trickle

possível encontrar as velocidades médias de *upload*, portanto foi assumido que as taxas de upload são de 10% das taxas de download (relação utilizada pelas operadoras de Internet brasileiras).

Em cada um dos casos de teste realizados também foram comparados os impactos da redundância dos resultados das atividades. Para verificar a veracidade de um resultado o sistema pode exigir que a mesma atividade seja executada n vezes. Para os testes com redundância, exigiu-se que cada resultado seja produzido apenas duas vezes.

Todos os casos de teste apresentados foram comparados em três diferentes situações. Na primeira, a execução é totalmente local, utilizando o computador 1, o computador mais potente entre os disponíveis, e sendo processada uma atividade de cada vez. A segunda e a terceira situações são, respectivamente, a computação voluntária e a solução aplicando técnicas de P2P. Ambas utilizando o computador 4 como servidor e os demais como voluntários. Em todos os testes nunca foi solicitado a um voluntário que executasse mais de uma atividade ao mesmo tempo (no máximo o voluntário poderia estar executando uma atividade enquanto baixa parâmetros de outro). Apesar de ser facilmente configurável no sistema a solicitação de execução de várias atividades simultâneas nos voluntários, optou-se por não fazer isto para diminuir o número de variáveis evolvidas nos testes.

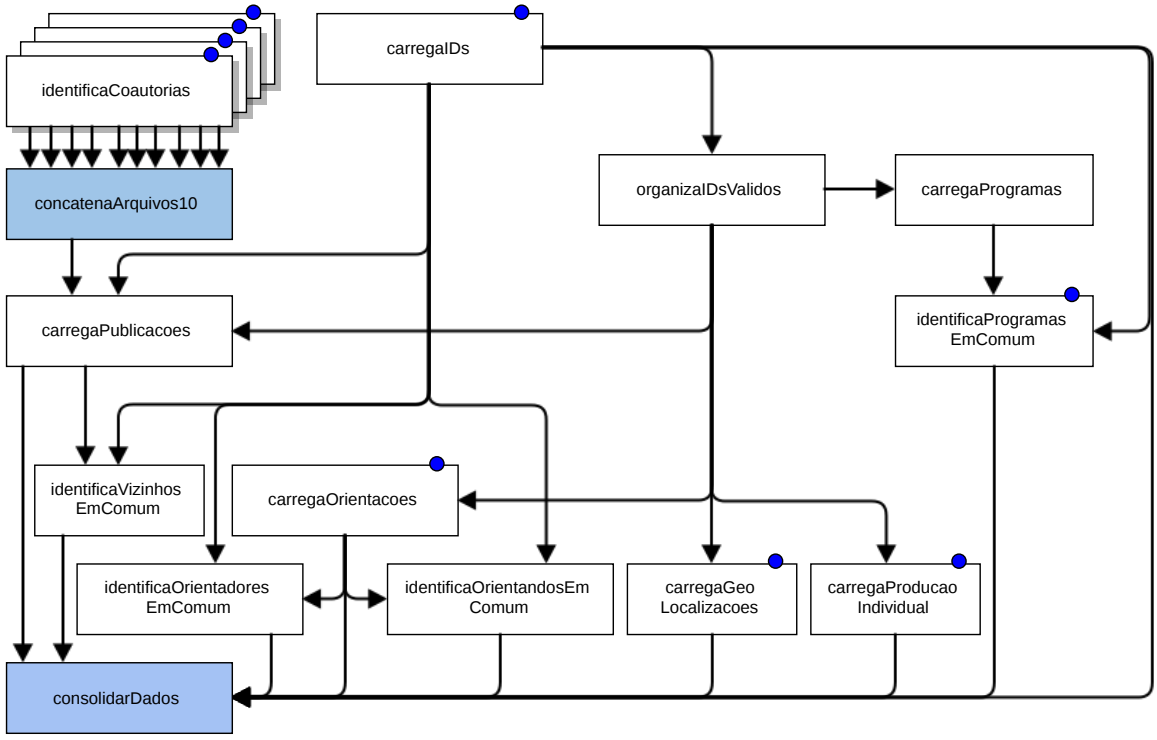
As próximas subseções descrevem os dois casos de testes utilizados.

5.1.1 Análise de Redes Sociais

O primeiro caso de teste é baseado em um workflow real de análise de redes sociais (DIGIAMPIETRI; SANTIAGO; ALVES, 2013; DIGIAMPIETRI et al., 2014a) e é apresentado, como pode ser visto na figura 8. Antes de mais nada dois item devem ser esclarecidos. Primeiro, os pequenos círculos azuis sobre as atividades representam parâmetros que não são saídas de nenhuma atividade, são dados pelo projetista do workflow e estão disponíveis desde o início da execução. Segundo, as atividades com fundo azul podem ser executadas localmente independente de qual método de escalonamento seja escolhido, isto é, mesmo que os testes sejam realizados utilizando a computação voluntária as atividades podem ser processados localmente, e neste caso serão testadas e analisadas as duas condições (com e sem a execução local destas atividades). No SGW desenvolvido, ficou a cargo do projetista determinar se uma atividade poderá ou não ser utilizada no servidor. Na versão atual do sistema, por padrão, as atividades são configuradas para

não serem executadas no servidor visando a não onerar o processamento do servidor, especialmente imaginando situações nas quais muitos workflows estariam executando ao mesmo tempo. Porém, como dito, o projetista pode habilitar a execução de atividades específicas no servidor.

Figura 8 – Workflow de teste baseado em um problema real de análise de redes sociais



Fonte: Caio Rafael do Nascimento Santiago, 2015

O objetivo do workflow da figura 8 é produzir um conjunto de atributos ou características para serem utilizados no processo de predição de coautorias em redes acadêmicas (DIGIAMPIETRI; SANTIAGO; ALVES, 2013; DIGIAMPIETRI et al., 2014a). Estas características são calculadas para cada par de pesquisadores e serão utilizadas para tentar predizer se um dado par de pesquisadores irá ou não colaborar na publicação de um artigo. Para isto são utilizadas diferentes informações oriundas dos currículos da plataforma Lattes (por exemplo, lista de publicações de cada pesquisador, lista de orientações, endereço profissional). Os dados utilizados nos testes são de 657 pesquisadores que são orientadores permanentes em programas de pós-graduação na área de Ciência da Computação. De maneira resumida, o workflow funciona da seguinte maneira. A maioria das atividades é responsável por produzir os dados referentes a alguma das características que será utilizada

posteriormente (fora do presente workflow) para a predição de coautorias. Por exemplo, as dez instâncias da atividade *identificaCoautorias* são chamadas cada uma recebendo um arquivo de entrada que corresponde a um subconjunto da lista total de publicações de cada um dos pesquisadores, os resultados destas atividades são concatenados pela atividade *concatenaArquivos10*. O resultado é utilizado para identificar a característica número de vizinhos em comuns pela atividade *identificaVizinhosEmComum* (considerando uma rede de coautorias). Outra característica que é calculada é se o par de pesquisadores possui orientandos em comum (atividade *identificaOrientandosEmComum*). Por fim, as diversas características calculadas são agrupadas pela atividade *consolidarDados*. O resultado desta atividade pode ser lido utilizado como entrada para um algoritmo de predição (DIGIAMPIETRI; SANTIAGO; ALVES, 2013).

Os parâmetros das 10 atividades *identificaCoautorias* são os que possuem maior volume de dados (alguns MBs), sendo que o maior possui 6,5 MBs. O restante dos parâmetros fornecidos pelo projetista também possuem um maior volume de dados do que seus derivados (os resultados parciais das atividades). Os dados trafegados são relativamente pequenos, mas possuem tamanho suficiente para, em situações de alta concorrência e/ou baixa velocidade, causar congestionamento na rede. Já as classes Java (cujas funções correspondem às atividades) são bastante pequenas, possuindo no máximo 20 KBs.

Cada caso foi executado três vezes, pois a diferença de desempenho entre os computadores e a diferença de processamento entre as atividades afeta a questão do balanceamento dos computadores, o que dificulta a verificação de resultados únicos. Portanto os resultados apresentados correspondem aos valores médios dessas execuções.

5.1.2 Exemplos Simplificados

Foram realizados outros testes baseados nas estruturas básicas de Workflows apresentadas em (BHARATHI et al., 2008) e discutida na seção 2.6. Ao todo são quatro estruturas: *Data Aggregation*, *Data Distribution*, emphData Redistribution e *Pipeline*.

Cada estrutura foi testada isoladamente, isso é, cada caso de teste se limitou a um único tipo de estrutura, e todas utilizaram a mesma forma com exatamente o mesmo número de atividades representadas na figura 1, exibida na seção 2.6.

Duas situações foram analisadas: na primeira os testes foram formados com apenas uma estrutura; na segunda os teste continham um conjunto de 10 estruturas iguais, mas

independentes entre si (por exemplo, para o teste do *Pipeline* ao invés de um workflow com uma única sequência de atividades, foi também criado um workflow com 10 sequências destas atividades). O objetivo destas duas situações é verificar o impacto de atividades dependentes quando os voluntários podem ou não processar outras atividades disponíveis.

As atividades utilizadas nestes casos de teste foram atividades de exemplo, que realizam um mesmo processamento (apenas para simular um processamento intenso). O processamento realizado por elas faz a verificação do problema clássico $3n+1$ (LAGARIAS, 1985) para o intervalo de 1 a 500.000.000, portanto a diferença nos tempos de execução de cada instância da atividade se deve apenas às diferenças de capacidade dos computadores. Além disso, os dados de entrada e saída também foram definidos com tamanhos constantes, de 2 MBs (vale destacar que a atividade utilizada nestes testes, na prática, não precisaria de entrada [por estarmos usando um intervalo constante de valores] e a saída poderia ser apenas um *verdadeiro* ou *falso* indicando que para o intervalo fixo utilizado a conjectura foi ou não confirmada, porém, estes parâmetros com tamanho fixo foram utilizados para simular situações reais onde dados de entrada são necessários antes da execução da atividade a qual produz dados de saída que serão utilizados pela atividade seguinte ou correspondem ao resultado final da execução do workflow).

5.2 Resultados

As subseções a seguir descrevem separadamente os resultados obtidos.

5.2.1 Análise de Rede Sociais

Os resultados da aplicação de técnicas de computação P2P foram bastante positivos. As execuções foram mais rápidas quando comparadas às outras soluções, como pode ser visto na tabela 4.

Existem dois motivos principais para os tempos terem diminuído. O primeiro motivo é que não houve aumento de tempo de execução das atividades causado pelo processamento que acontece em segundo plano do processamento da atividade (foram comparados os tempo do computador 1 entre os 3 sistemas e não foram constatadas diferenças), não há concorrência por recursos, pois o processamento em segundo plano é bastante simples e se aproveita da existência de mais de um núcleo no processador para não concorrer com

Tabela 4 – Tempo de duração do processamento dos casos de testes reais de análise de redes sociais

Execução		Duração (tempo)
Local		7:27:19
Computação Voluntária	Com execução local	04:36:31
	Sem execução local	04:57:00
Computação Voluntária com P2P	Com execução local	04:22:19
	Sem execução local	04:27:45

Fonte: Caio Rafael do Nascimento Santiago, 2015

o processamento da atividade em execução. O segundo motivo é que o comportamento proativo do escalonamento local prepara as atividades para serem processadas enquanto se processa a atividade anterior. A maioria das atividades já estavam prontas para serem processadas no momento em que foram escolhidas pelo escalonador local, reduzindo o tempo que o cliente demora baixando parâmetros antes de iniciar o processamento da atividade.

A tabela 5 apresenta o tempo médio gasto pelos voluntários apenas com transferências de dados, isto é, sem que o voluntário estivesse processando algum atividade (não foi considerado o tempo ocioso quando não havia atividades disponíveis para se processar). Na média, este tempo correspondeu a 26,7% do tempo total de execução para o uso de computação voluntária tradicional e 13,8% para o caso da computação voluntário utilizando P2P. Os valores da computação voluntária com P2P refletem, em grande parte, o tempo que o voluntário leva para baixar os parâmetros da primeira atividade que irá executar, já que no momento que as atividades subsequentes estão disponíveis para a execução, o voluntário já baixou a maior parte de seus parâmetros. O menor tempo de espera, por exemplo, no caso do *Pipeline* é justificada pelo fato de, na solução implementada, se o computador voluntário terminou a execução de uma atividade e continua disponível para a execução de outras ele já possui a saída da atividade que acabou de executar e então está pronto para executar a atividade subsequente (a próxima do *Pipeline* não precisando assim gastar tempo copiando parâmetros). Para o caso da computação voluntária tradicional, o voluntário devolverá primeiro a saída para o servidor e só depois se voluntariará para a execução de novas atividades.

Os voluntários acabam baixando mais parâmetros do que o necessário, isso porque não há como saber no momento do *donwload* se a atividade ainda estará disponível

Tabela 5 – Tempo médio de espera transferindo dados nos voluntários nos casos de teste de análise de redes sociais

Execução		Duração (tempo)
Computação Voluntária	Com execução local	00:26:07
	Sem execução local	00:28:19
Computação Voluntária com P2P	Com execução local	00:18:27
	Sem execução local	00:19:48

Fonte: Caio Rafael do Nascimento Santiago, 2015

para ser processada quando o voluntário for efetivamente escolher a próxima atividade para processar (a atividade pode já ter sido concluída ou já ter voluntários o suficiente a processando). Neste caso de teste praticamente todos os parâmetros são enviados para todos os voluntários. Sem a aplicação de técnicas de P2P o tráfego total seria aproximadamente três vezes maior que a computação voluntária, se for considerado apenas o tráfego de *upload* do servidor. Porém o total apresentou menos que o dobro da computação voluntária, como pode-se verificar na tabela 6. Nota-se que nos casos em que os parâmetros são reutilizados são aqueles em que os voluntários executam as atividades azuis (Execução Local), não há um crescimento muito maior na transferência de dados.

Tabela 6 – Dados transferidos por cada voluntário em MBs no caso real de redes sociais

Execução		Servidor		Voluntário 1		Voluntário 2		Voluntário 3	
		Up.	Down.	Up.	Down.	Up.	Down.	Up.	Down.
CV	Local	64,29	13,28	5,79	24,21	3,45	22,40	3,94	21,25
	Sem local	75,54	15,01	4,14	31,63	5,59	25,04	5,66	23,21
CV com P2P	Local	91,95	15,25	4,91	28,91	7,20	27,69	2,98	33,10
	Sem local	115,74	17,92	6,30	38,29	6,74	37,70	5,00	38,82

Fonte: Caio Rafael do Nascimento Santiago, 2015

A situação dos dados transferidos entre computadores se inverte quando consideramos a redundância de resultados, uma situação muito comum em sistemas de computação voluntária. Cada atividade foi processada duas vezes (utilizando-se voluntários diferentes) e os casos de teste utilizados os que não executavam nenhuma atividade no servidor. A computação voluntária demorou pouco mais de oito horas e meia (08:33:30) e a aplicação de P2P 08:15:02 (representando uma economia de tempo de apenas 3,5%). Porém, quanto a questão de dados transferidos (Tabela 7) pode-se notar que os valores para a

computação voluntária praticamente dobrou (quando comparados com os testes sem redundância) enquanto que para a aplicação de P2P estes valores se mantiveram praticamente inalterados.

Tabela 7 – Dados transferidos por cada voluntário em MBs no caso real de redes sociais sem a execução de atividades no servidor (com redundância)

Execução	Servidor		Voluntário 1		Voluntário 2		Voluntário 3	
	Up.	Down.	Up.	Down.	Up.	Down.	Up.	Down.
CV	153,31	30,18	9,41	64,15	11,26	54,65	10,22	38,38
CV com P2P	99,39	31,33	7,12	23,71	14,76	42,62	10,17	29,85

Fonte: Caio Rafael do Nascimento Santiago, 2015

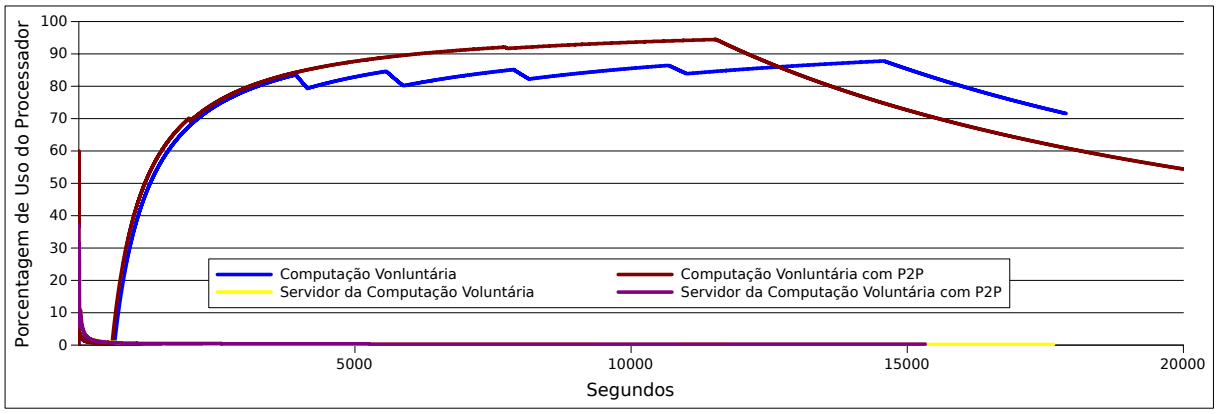
É possível também observar que a aplicação de P2P resultou em um melhor aproveitamento do processamento, a figura 9 apresenta a porcentagem de processamento em um caso de teste com o voluntário 1 e o servidor (vale lembrar que os testes foram realizados com três voluntários, mas para facilitar a leitura da figura 9 foram representados apenas a porcentagem de uso de processamento do servidor e de um voluntário já que o comportamento dos voluntários foi bastante semelhante). Para os dois casos (computação voluntária e computação voluntária com P2P) o servidor manteve, na média, baixíssimas taxas de processamento, abrindo a possibilidades de aproveitar o servidor para também executar atividades assim como os voluntários, ou realizar qualquer outro tipo de processamento, ou mesmo hospedar outros projetos de computação voluntária. Quanto aos voluntários existem dois comportamentos: a computação voluntária tradicional apresentou uma oscilação na curva de processamento, as "quedas" na curva representam os pontos em que uma atividade concluiu, baixou os parâmetros necessários, e começou o processamento da próxima atividade; e no caso com P2P não há essas "quebras" na curva já que o tempo entre as atividades foi minimizado.

O uso de memória foi bastante similar entre as duas soluções. Os servidores da computação voluntária sem e com a aplicação de P2P utilizaram até, respectivamente, 4,9% e 7,4%. Enquanto que os voluntários utilizaram até 9% e 7,4%, respectivamente.

5.2.2 Exemplos Simplificados

Inicialmente, serão analisados os resultados dos testes contendo apenas uma estrutura. Os tempos de execução (Tabela 8) mostram que nestas condições o resultados

Figura 9 – Utilização do processamento dos testes baseados em um problema real de análise de redes sociais



Fonte: Caio Rafael do Nascimento Santiago, 2015

da computação local obteve os melhores resultados para todos os casos, seguida pela solução desenvolvida neste trabalho, e a computação voluntária tradicional com os piores resultados (exceto na estrutura *Data Distribution* no qual o tempo das duas últimas soluções foi bastante parecido). Existe uma combinação de fatores que corrobora para esta situação: o computador que realiza a computação local é o mais rápido entre os três; tanto a computação voluntária tradicional quanto a modificação proposta neste trabalho são fortemente dependentes da existência de atividades paralelas, que não estão massivamente presentes neste caso. Em particular o número relativamente pequeno de atividades (algumas podendo ser processadas de maneira paralela e outras não), combinado com os custos relativamente altos de transferência de dados entre o servidor e os voluntários (de acordo com as configurações de taxas de transmissão de dados apresentadas no início deste capítulo) e, como já apresentado, nos testes de execução local, foi utilizado o melhor computador para executar todas as atividades.

A diferença de capacidade de processamento entre os computadores causou uma situação na qual, em alguns casos, computadores mais lentos estão processando atividades enquanto que os computadores mais rápidos, e portanto mais aptos, estão ociosos. Quando não há muitas atividades paralelas o tempo total da execução pode ficar dependente do tempo de execução pelo computador mais lento, prejudicando o desempenho total do sistema. A exposição deste caso pode ser vista na figura 10 que mostra quando os computadores estão ativos ou ociosos em função do tempo.

Tabela 8 – Tempo de duração do processamento dos casos de testes exemplos simplificados (com uma estrutura)

Estrutura	Execução	Tempo
Data Agregation	Local	0:21:55
	CV	0:39:04
	P2P	0:38:06
Data Distribution	Local	0:21:37
	CV	0:25:43
	P2P	0:25:59
Data Redistribution	Local	0:39:33
	CV	0:58:48
	P2P	0:50:07
Pipeline	Local	0:15:19
	CV	0:37:31
	P2P	0:20:18

Fonte: Caio Rafael do Nascimento Santiago, 2015

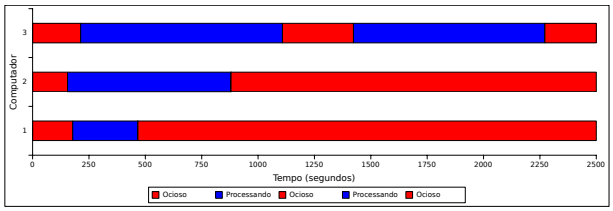
Esta situação não se beneficia dos pontos positivos da solução desenvolvida (em relação a solução local), mas mesmo nesta situação não favorável a solução proposta apresentou melhoras significativas na maiorias dos casos em relação à computação voluntária tradicional. O algoritmo de escalonamento proposto baixa outros parâmetros para processar as próximas atividades e, portanto, a primeira atividade nunca se beneficia desta estratégia. Além disso, como pode-se notar na figura 10 nestes testes há várias situações em que os voluntários estão processando uma atividade e não há mais nenhum parâmetro disponível para ser baixado.

Quanto a questão de tráfego de dados, os resultados foram muito próximos, como pode-se ver na tabela 9. Novamente a questão de não haver parâmetros disponíveis para baixar no momento de processamento influenciou este caso. Os resultados do *Data Distribution* e do *Pipeline* foram bastante próximos. O *Data Aggregation* e o *Data Redistribution* foram melhores, os resultados são modestos, mas proporcionalmente são resultados relevantes.

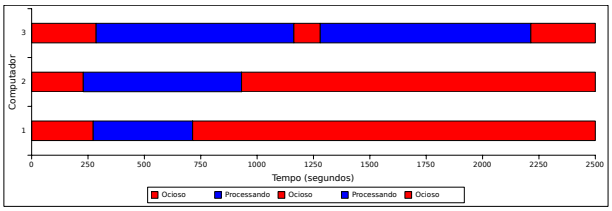
Já nos caso em que os teste continham um conjunto de 10 estruturas iguais e independentes os resultados foram mais promissores e serão discutidos a seguir. Notem que este tipo de caso de teste ajuda a ilustrar o comportamento do sistema imaginando-se tanto a execução de workflows mais complexos, mas também como o sistema se comportaria no casa da execução de 10 workflows (simples) ao mesmo tempo.

Figura 10 – Processamento em função do tempo (segundos) de cada voluntário no caso de exemplos simplificados (com uma estrutura)

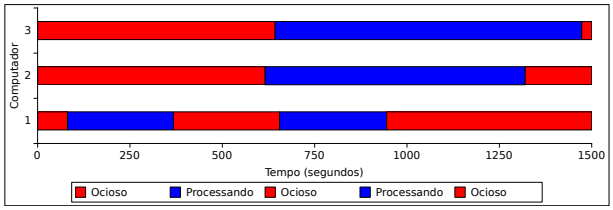
(a) Data Aggregation em CV com P2P



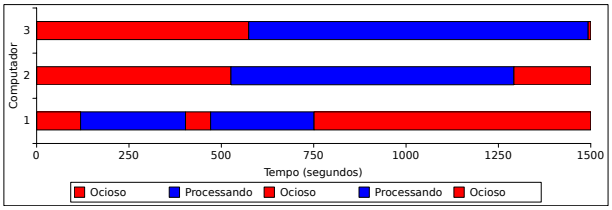
(b) Data Aggregation em CV com P2P



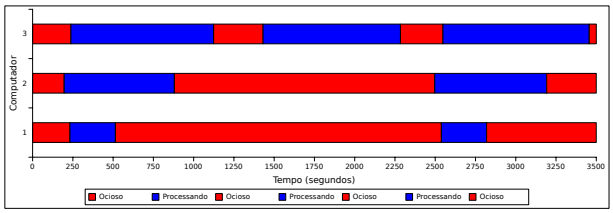
(c) Data Distribution em CV



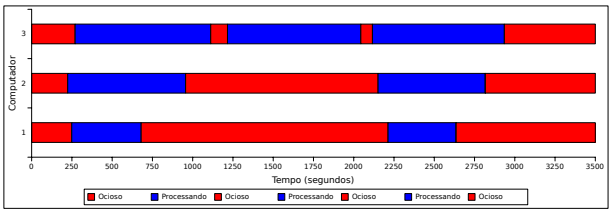
(d) Data Distribution em CV com P2P



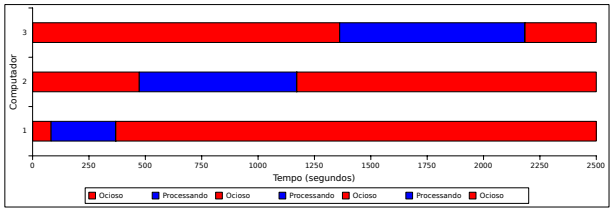
(e) Data Redistribution em CV com P2P



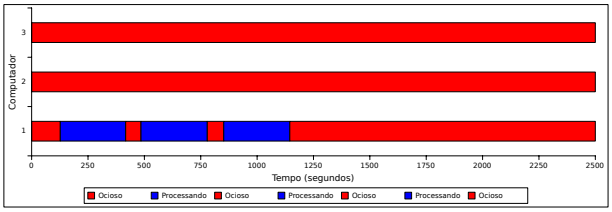
(f) Data Redistribution em CV com P2P



(g) Pipeline em CV



(h) Pipeline em CV com P2P



Fonte: Caio Rafael do Nascimento Santiago, 2015

Assim como nos casos de teste utilizando o *workflow* de análise de redes sociais, para os exemplos simplificados com conjunto de estruturas (ver subseção 5.1.2) a computação voluntária com P2P se mostrou mais rápida, os tempos estão presentes na tabela 10. Pode-se notar que os tempos foram consideravelmente mais baixos, mas como todas as atividades são iguais os tempos são mais constantes (do que no estudo de caso real). A comparação desta relação de tempo entre a computação voluntária tradicional e utilizando computação P2P (descrita na tabela 11) mostra uma variação de 75% a 87% e, novamente, essa variação é causada pela diminuição do tempo para se obter os parâmetros.

Pode-se notar pela tabela 12 que o tempo que os clientes gastaram apenas transferindo dados foi bastante reduzido, menos da metade em todos os casos. O gasto de

Tabela 9 – Dados transferidos por cada voluntário em MBs no caso de exemplos simplificados (com uma estrutura)

Execução		Servidor		Voluntário 1		Voluntário 2		Voluntário 3	
		Up.	Down.	Up.	Down.	Up.	Down.	Up.	Down.
Data Agregation	CV	12,57	8,60	2,04	2,00	2,12	2,22	4,34	8,61
	P2P	10,43	8,51	2,30	6,12	2,21	5,10	3,96	2,86
Data Distribution	CV	8,49	8,35	3,75	3,78	2,14	2,15	2,11	2,18
	P2P	8,63	8,38	3,77	2,07	2,20	3,13	2,15	3,22
Data Redistribution	CV	17,72	14,25	4,17	4,10	4,35	4,41	6,44	10,87
	P2P	14,41	14,63	4,15	6,70	4,50	4,58	2,78	2,88
Pipeline	CV	6,17	6,23	1,82	1,84	2,11	2,15	2,13	2,25
	P2P	6,57	6,22	5,69	2,03	0,11	2,15	0,12	2,26

Fonte: Caio Rafael do Nascimento Santiago, 2015

Tabela 10 – Tempo de duração do processamento dos casos de testes exemplos simplificados (com 10 estruturas)

Estrutura	Execução	Tempo
Data Agregation	Local	3:49:56
	CV	3:09:50
	P2P	2:22:37
Data Distribution	Local	3:06:35
	CV	2:51:00
	P2P	2:29:13
Data Redistribution	Local	6:50:19
	CV	5:04:07
	P2P	4:12:43
Pipeline	Local	2:20:45
	CV	2:20:37
	P2P	1:48:28

Fonte: Caio Rafael do Nascimento Santiago, 2015

tempo da computação voluntária com P2P se deve principalmente por causa do envio da resposta para o servidor, pois no momento de enviar a resposta para o servidor o cliente não realiza nenhuma outra tarefa. O efeito dessa redução de tempo com transferência de dados impacta diretamente no tempo total de execução dos casos, já que se foram descontados esse tempo sobre o tempo total de execução os resultados se aproximam.

Quanto às questões do tráfego de dados os resultados foram bastante positivos, diferente do caso de teste em redes sociais. Pode-se notar na tabela 13 que a aplicação de P2P obteve resultados consideravelmente melhores, quando consideramos apenas o

Tabela 11 – Comparação entre o tempo de CV com o tempo da CV com P2P (com 10 estruturas)

Estrutura	Porcentagem de tempo de execução da CV pela P2P
Data Aggregation	75%
Data Distribution	87%
Data Redistribution	83%
Pipeline	77%

Fonte: Caio Rafael do Nascimento Santiago, 2015

Tabela 12 – Tempo médio de espera transferindo dados pelos voluntários nos casos de exemplos simplificados (com 10 estruturas)

Estrutura	Execução	Tempo
Data Aggregation	CV	00:57:55
	P2P	00:21:53
Data Distribution	CV	00:41:37
	P2P	00:20:04
Data Redistribution	CV	01:25:21
	P2P	00:31:49
Pipeline	CV	00:33:24
	P2P	00:15:09

Fonte: Caio Rafael do Nascimento Santiago, 2015

servidor que é justamente o foco desta abordagem. Quase todos os resultados obtiveram um total de *upload* abaixo da metade da computação voluntária tradicional e o principal motivo para a diferença entre os dois casos de testes é que no caso de análise de redes sociais as atividades possuíam saídas consideravelmente menores que suas entradas, já nestes casos ambas possuem o mesmo tamanho, o que favoreceu essa abordagem. Quanto ao total de *download* os valores foram muito próximos, mas a solução com P2P causou um cenário onde o servidor recebeu mais dados do que enviou. Receber mais do que enviar é tipicamente considerada uma coisa bastante positiva em muitos cenários reais (por exemplo, na estrutura da internet para usuários comuns, as operadoras garantam taxas de download bem maiores que as de upload), favorecendo novamente a abordagem proposta.

Na situação em que são exigidas duas execuções da mesma atividade (redundância), os resultados se mantêm bastante próximos aos casos de teste de rede sociais. Conforme pode ser observado na tabela 14, os tempos de execução foram consideravelmente mais

Tabela 13 – Dados transferidos por cada voluntário em MBs no caso de exemplos simplificados (com 10 estruturas)

Execução		Servidor		Voluntário 1		Voluntário 2		Voluntário 3	
		Up.	Down.	Up.	Down.	Up.	Down.	Up.	Down.
Data Aggregation	CV	120,68	82,38	37,27	60,17	25,66	30,57	20,74	34,10
	P2P	75,92	85,04	40,20	18,73	22,98	26,14	16,49	26,22
Data Distribution	CV	84,64	84,28	37,48	37,57	25,72	26,63	18,37	19,14
	P2P	33,97	83,37	38,37	11,16	25,41	13,00	18,31	8,66
Data Redistribution	CV	185,21	147,01	62,49	92,37	44,27	43,99	36,20	45,41
	P2P	71,02	143,72	63,14	19,98	42,48	24,51	34,86	27,28
Pipeline	CV	61,79	61,29	26,85	26,75	18,91	19,41	14,21	15,25
	P2P	28,13	59,52	30,32	8,36	14,46	8,57	14,35	10,51

Fonte: Caio Rafael do Nascimento Santiago, 2015

rápidos, em uma proporção similar a que foi encontrada no mesmo caso sem redundância, como pode-se notar na tabela 15.

Tabela 14 – Tempo de duração do processamento dos casos de testes exemplos simplificados (com 10 estruturas e redundância)

Estrutura	Execução	Tempo
Data Agregation	CV	7:17:01
	P2P	5:56:38
Data Distribution	CV	6:58:02
	P2P	5:13:22
Data Redistribution	CV	11:14:16
	P2P	8:44:02
Pipeline	CV	5:13:39
	P2P	4:15:22

Fonte: Caio Rafael do Nascimento Santiago, 2015

Assim como no caso sem redundância, a banda utilizada pelo servidor também foi menor com a computação voluntária com P2P (Tabela 16) e o aumento da banda de *upload* foi relativamente baixo ao se comparar com o da computação voluntária tradicional (este último foi de aproximadamente 100% em relação a situação sem redundância).

Tabela 15 – Comparação entre o tempo de CV com o tempo da CV com P2P (com e sem redundância – 10 estruturas)

Estrutura	Porcentagem de tempo de execução da CV pela P2P	
	Sem redundância	Com redundância
Data Agregation	75%	82%
Data Distribution	87%	75%
Data Redistribution	83%	78%
Pipeline	77%	81%

Fonte: Caio Rafael do Nascimento Santiago, 2015

Tabela 16 – Dados transferidos por cada voluntário em MBs no caso de exemplos simplificados (com 10 estruturas e redundância)

Execução		Servidor		Voluntário 1		Voluntário 2		Voluntário 3	
		Up.	Down.	Up.	Down.	Up.	Down.	Up.	Down.
Data Aggregation	CV	243,28	167,59	75,72	113,37	49,73	70,50	43,52	66,35
	P2P	80,29	165,71	65,30	20,80	44,60	23,73	44,99	31,59
Data Distribution	CV	168,12	165,18	76,21	75,33	46,86	47,34	41,01	44,19
	P2P	35,77	164,07	69,75	10,32	46,21	11,90	42,00	11,75
Data Redistribution	CV	372,88	293,27	132,15	168,38	81,50	98,83	71,88	101,33
	P2P	86,46	288,25	127,12	28,60	80,16	23,84	70,74	31,54
Pipeline	CV	121,99	124,28	55,21	54,73	33,55	33,93	32,48	34,27
	P2P	26,21	123,21	58,20	7,97	35,74	8,24	26,36	9,40

Fonte: Caio Rafael do Nascimento Santiago, 2015

5.3 Considerações Finais

A solução desenvolvida neste trabalho e avaliada neste capítulo se mostrou bastante promissora, quando confrontada com os outros dois métodos de escalonamento: a computação voluntária tradicional e a execução local. Os resultados foram positivos, nos dois conjuntos de teste utilizados, mostrando uma melhora real no desempenho.

O primeiro caso de teste utilizou um *workflow* real da área de análise de redes sociais. A solução se mostrou mais rápida do que os outros dois métodos de escalonamento, tanto com e sem redundância. A banda gasta pelo servidor registrou uma considerável piora relativa na situação sem redundância, o principal motivo para isso acontecer se deve ao fato dos tráfegos iniciais serem maiores que os dados resultantes da execução das atividades. Na situação com redundância a questão do gasto de banda se inverte e a solução desenvolvida se torna mais atraente.

O segundo conjunto de teste utilizou exemplos simplificados (*toy example*) das estruturas básicas de workflows. Todos os quesitos verificados foram positivos para a computação voluntária com P2P em relação à computação voluntária tradicional. O tempo de execução foi de 12 a 25% mais rápido do que a computação voluntária tradicional. Também foi reduzida as taxa de *upload* do servidor em ambos os casos, isto é, com e sem redundância, mas na situação com redundância as melhorias de desempenho são mais expressivas.

Em linhas gerais e exceto para a execução de workflows muito simples, a solução desenvolvida apresentou uma melhora no desempenho, o tempo de execução foi reduzido, o tempo que os voluntários ficam inativos (quando não estão processando atividades) foi reduzido, e nos casos em que o tamanho dos parâmetros das atividades não estão desequilibrados (comparando-se os tamanhos das entradas e das saídas) ou quando há redundância de atividades o gasto de banda do servidor também foi reduzido.

6 CONCLUSÕES

Este trabalho se propôs a especificar, desenvolver e testar uma solução para a execução distribuída de workflows combinando computação voluntária e técnicas de computação ponto-a-ponto.

A principal ideia do projeto foi tentar aproveitar o momento em que os voluntários estavam com o processador ocupado executando atividades, mas inativos do ponto de vista de tráfego de dados para preparar as próximas atividades a serem processadas, bem como para auxiliar outros voluntários compartilhando parâmetros. Essa abordagem diminuiu o tempo em que o voluntário não está processando atividades (por estar copiando parâmetros) e diminui a exigência sobre a banda do servidor, sem aumentar o tempo que levaria para uma atividade ser processada em relação a uma abordagem convencional de computação voluntária.

A fim de atingir o objetivo proposto foi realizada uma revisão sistemática acerca de métodos de escalonamento de ambientes de computação voluntária que apliquem técnicas de computação P2P. A revisão encontrou poucos trabalhos relacionados com o tema, corroborando com a contribuição potencial deste trabalho. Os trabalhos foram analisados e suas características foram apontadas.

Em seguida foi especificado e desenvolvido um Sistema de Gerenciamento de Workflows (SGW) a partir de um sistema já existente, construído no decorrer do projeto Jovem Pesquisador em Centros Emergentes – FAPESP do orientador. O SGW foi reformulado e expandido para torná-lo mais modularizado e flexível o suficiente para utilizar diferentes tipos de técnicas de escalonamento. Neste trabalho formam desenvolvidos três métodos de escalonamento: considerando execução local, execução com computação voluntária tradicional e, a solução proposta, considerando computação voluntária utilizando técnicas P2P.

A solução proposta foi comparada com a execução local de atividades e com a computação voluntária tradicional. Os três métodos foram testados em dois cenários de testes, e a partir das métricas sobre os resultados obtidos foi constada a melhora de desempenho da solução proposta. Em ambos os cenários de teste, o caso real de análise de redes sociais e os conjuntos de exemplos simplificados de estruturas de workflows, foi possível obter métricas referentes ao tempo de execução, de transmissão de dados, de

espera sem processamento e métricas referentes ao gasto/utilização de banda do servidor e dos voluntários. São resultados bastante positivos que mostram que a solução é capaz de diminuir o tempo total de uma execução e, em alguns casos, diminuir o gasto de banda do servidor.

A seguir, são apresentadas as principais contribuições deste trabalho e as propostas de trabalhos futuros.

6.1 Principais Contribuições

Este trabalho teve como principal objetivo e contribuição, a construção de um ambiente de computação voluntária funcional que utilizasse técnicas de computação P2P em seu escalonamento. O objetivo foi alcançado uma vez que o ambiente foi construído e obteve resultados superiores aos da computação voluntária tradicional nas questões da duração total da execução dos *workflows* e na exigência de largura de banda que foram ambos diminuídos.

Além do objetivo principal deste trabalho também foram alcançadas outras contribuições, são elas:

- Uma revisão sistemática foi realizada e disponibilizada acerca do tema: métodos de escalonamento de ambientes de computação voluntária que utilizam técnicas de P2P. Esta revisão pode servir de base para trabalhos futuros correlatos.
- Foi construído um SGW correspondendo à extensão do SGW desenvolvido no Jovem Pesquisador em Centros Emergentes – FAPESP do orientador. O SGW foi original restruturado, tornando-o mais organizado e modularizado. O SGW resultante é totalmente funcional e possui três métodos de escalonamento implementados atualmente e pode ser facilmente ser modificado e expandido para trabalhar com outros métodos.
- Também foi construído um ambiente de computação voluntária totalmente funcional. O ambiente é capaz de executar remotamente, em computadores pessoais, funções desenvolvidas em Java. O ambiente se encarrega de enviar as classes e parâmetros necessários, tornando o trabalho do projetista do workflow bastante transparente. Apesar de não ser o foco deste trabalho, os serviços Web continuam podendo ser executados por atividades dos *worklfows*, pois, assim como na versão original do SGW,

estes serviços são encapsulados por funções desenvolvidas em Java. Obviamente, a execução destes serviços não onera o processador do servidor ou do voluntário, porém, dependendo do volume de dados dos resultados intermediários das atividades, a solução proposta pode ser útil para uma melhor distribuição de banda.

- O código fonte da solução desenvolvida, bem como dos testes executados, está disponível na web[1] para poder ser utilizado tanto para execução de experimentos científicos, mas também como base para trabalhos futuros.
- Até o momento, a pesquisa deste mestrado foi parcialmente utilizada na publicação de quatro artigos. Três deles relacionados a partes da solução desenvolvida (DIGIAMPIETRI et al., 2013; DIGIAMPIETRI et al., 2014b; DIGIAMPIETRI et al., 2014) e um relacionado ao estudo de caso (DIGIAMPIETRI; SANTIAGO; ALVES, 2013).

6.2 Trabalhos Futuros

Durante o desenvolvimento do trabalho, foram identificadas diversas possibilidades de continuidade e evolução do ambiente desenvolvido, são elas:

1. Uma possível evolução deste trabalho seria abordar ambientes ainda mais heterogêneos, considerando os diferentes elementos, capacidades de processamento, velocidades de transmissão e graus de confiança dos resultados. Um exemplo deste conceito é a junção de diferentes tipos de *workers* no mesmo ambiente, por exemplo, como a junção de computadores voluntários e *grids* de diferentes naturezas.
2. A comunicação entre os voluntários, atualmente, não está pronta para diferentes tipos de conexões. Por exemplo, computadores com diferentes níveis de profundidade em rede NAT (*Network Address Translation*) podem encontrar dificuldade de se comunicarem. Este mecanismo, portanto, pode ser aprimorado para utilizar uma solução mais sofisticada e versátil.
3. Os testes realizados utilizaram poucos computadores, não sendo possível assim saber como o método de escalonamento vai se comportar quando houver centenas ou milhares de voluntários. Assim, testes utilizando uma infraestrutura maior são desejáveis.

[1] http://www.each.usp.br/digiampietri/codigos/codigos_crns.7z

4. Também não foi testado o que acontece quando existe uma quantidade massiva de atividades, e nem quando muitos *workflows* são executados ao mesmo tempo. Apesar da solução desenvolvida apresentar, relativamente, um desempenho melhor do que a computação voluntária convencional nos exemplos em que um maior número de atividades foi executado, é interessante avaliar isto em exemplos ainda maiores.
5. Outra questão não verificada é como o ambiente se comporta com o aparecimento de falhas. Uma futura contribuição seria elaborar uma gestão eficiente de falhas, fazendo, por exemplo, uma estimativa da redundância necessária para cada atividade a ser executada bem como uma avaliação das melhores formas de detecção e tratamento das falhas.
6. A versão atual da solução desenvolvida não solicita a um voluntário a execução paralela de mais de uma atividade. Uma contribuição futura seria possibilitar que computadores voluntários com CPUs multi-núcleos executem várias atividades simultaneamente.

AGRADECIMENTOS

O trabalho desenvolvido no decorrer deste mestrado foi financiado pela FAPESP (projetos 2013/07935-5 e 2009/10413-5).

REFERÊNCIAS

ACHARYA, A.; EDJLALI, G.; SALTZ, J. The utility of exploiting idle workstations for parallel computation. *ACM SIGMETRICS Performance Evaluation Review*, ACM, v. 25, n. 1, p. 225–234, jun. 1997. ISSN 01635999. Disponível em: <http://dl.acm.org/citation.cfm?id=258623.258691>. Citado na página 13.

AKAMAI. *The State of the Internet*. [S.l.], 2013. Citado na página 42.

ANDERSON, D. BOINC: a system for public-resource computing and storage. In: . [S.l.: s.n.], 2004. p. 4–10. Citado 4 vezes nas páginas 13, 18, 26 e 28.

ANDERSON, D. P. et al. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, ACM, v. 45, n. 11, p. 56–61, nov. 2002. ISSN 00010782. Disponível em: <http://dl.acm.org/ft_gateway.cfm?id=581573&type=html>. Citado na página 18.

ANDERSON, D. P.; FEDAK, G. The Computational and Storage Potential of Volunteer Computing. In: *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*. [S.l.: s.n.], 2006. v. 1, p. 73–80. Citado na página 13.

BARGA, R. S.; DIGIAMPIETRI, L. A. Automatic capture and efficient storage of e-Science experiment provenance. *Concurrency and Computation: Practice and Experience*, John Wiley & Sons, Ltd., v. 20, n. 5, p. 419–429, 2008. ISSN 1532-0634. Disponível em: <http://dx.doi.org/10.1002/cpe.1235>. Citado na página 21.

BEBERG, A. L. et al. Folding@home: Lessons from eight years of volunteer distributed computing. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009. p. 1–8. ISBN 978-1-4244-3751-1. ISSN 1530-2075. Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5160922>. Citado na página 18.

BHARATHI, S. et al. Characterization of scientific workflows. In: *2008 Third Workshop on Workflows in Support of Large-Scale Science*. IEEE, 2008. p. 1–10. ISBN 978-1-4244-2827-4. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4723958>. Citado 3 vezes nas páginas 15, 20 e 45.

CELAYA, J.; ARRONATEGUI, U. Distributed Scheduler of Workflows with Deadlines in a P2P Desktop Grid. In: *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010. p. 69–73. ISBN 978-1-4244-5672-7. ISSN 1066-6192. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5452507>. Citado 3 vezes nas páginas 25, 29 e 31.

DETHIER, G. et al. LBG-SQUARE Fault-Tolerant, Locality-Aware Co-Allocation in P2P Grids. In: *2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE, 2008. p. 252–258. ISBN 978-0-7695-3443-5. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4710988>. Citado 2 vezes nas páginas 13 e 18.

DIAS, J. et al. Improving Many-Task computing in scientific workflows using P2P techniques. In: *2010 3rd Workshop on Many-Task Computing on Grids and*

Supercomputers. IEEE, 2010. p. 1–10. ISBN 978-1-4244-9704-1. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5699430>. Citado 2 vezes nas páginas 13 e 18.

DIGIAMPIETRI, L. et al. Análise da rede dos doutores que atuam em computação no brasil. In: *CSBC-BraSNAM 2014*. [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 43 e 44.

DIGIAMPIETRI, L. et al. A framework for automatic composition of scientific experiments: Achievements, lessons learned and challenges. In: *CSBC 2014 - BreSci*. Brasília - DF: [s.n.], 2014. p. 315–322. Citado 2 vezes nas páginas 22 e 60.

DIGIAMPIETRI, L.; SANTIAGO, C.; ALVES, C. Predição de coautorias em redes sociais acadêmicas: um estudo exploratório em ciência da computação. In: *CSBC-BraSNAM 2013*. [S.l.: s.n.], 2013. Citado 4 vezes nas páginas 43, 44, 45 e 60.

DIGIAMPIETRI, L. et al. Um sistema de informação extensível para o reconhecimento automático de LIBRAS. In: *SBSI 2012 - Trilhas Técnicas (Technical Tracks)*. [S.l.: s.n.], 2012. p. 12. Citado na página 22.

DIGIAMPIETRI, L. A. et al. Combinando workflows e semântica para facilitar o reuso de software. *Revista de Informática Teórica e Aplicada: RITA*, v. 20, p. 73–89, 2013. Citado 3 vezes nas páginas 15, 22 e 60.

DIGIAMPIETRI, L. A. et al. An extensible framework for genomic and metagenomic analysis. In: *Advances in Bioinformatics and Computational Biology*. Springer International Publishing, 2014, (Lecture Notes in Computer Science, v. 8826). p. 1–8. ISBN 978-3-319-12417-9. Disponível em: <http://dx.doi.org/10.1007/978-3-319-12418-6_1>. Citado na página 60.

DIGIAMPIETRI, L. A.; PEREZ-ALCAZAR, J. d. J.; MEDEIROS, C. B. An ontology-based framework for bioinformatics workflows. *International Journal of Bioinformatics Research and Applications*, v. 3, n. 3, p. 268–285, 2007. Disponível em: <http://inderscience.metapress.com/content/BH83K31847158824>. Citado na página 21.

DIGIAMPIETRI, L. A. et al. Uso de Planejamento em Inteligência Artificial para o Desenvolvimento Automático de Software. In: *Autonomous Software Systems (AutoSoft 2011)*. [S.l.: s.n.], 2011. Citado 2 vezes nas páginas 15 e 22.

DORNEMANN, K. et al. Integrating Virtual Execution Environments into Peer-to-Peer Desktop Grids. In: *2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2012. p. 333–340. ISBN 978-1-4673-0226-5. ISSN 1066-6192. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6169569>. Citado 2 vezes nas páginas 13 e 18.

DUAN, K. et al. Composition of engineering web services with universal distributed data-flows framework based on ROA. In: *Proceedings of the Third International Workshop on RESTful Design - WS-REST '12*. New York, New York, USA: ACM Press, 2012. p. 41. ISBN 9781450311908. Disponível em: <http://dl.acm.org/citation.cfm?id=2307819.2307830>. Citado 4 vezes nas páginas 13, 14, 18 e 20.

JAGADISH, H. VBI-Tree: A Peer-to-Peer Framework for Supporting Multi-Dimensional Indexing Schemes. In: *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 2006. p. 34–34. ISBN 0-7695-2570-9. Disponível em: <http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1617402>. Citado na página 25.

KONDO, D. et al. Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems*, v. 23, n. 7, p. 888–903, 2007. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X06002111>. Citado na página 13.

KWAN, S. K.; JOGESH, K. M. Bag-of-Tasks applications scheduling on volunteer desktop grids with adaptive information dissemination. In: *IEEE Local Computer Network Conference*. IEEE, 2010. p. 544–551. ISBN 978-1-4244-8387-7. ISSN 0742-1303. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5735771>. Citado 3 vezes nas páginas 25, 29 e 30.

LAGARIAS, J. C. The 3x + 1 problem and its generalizations. *The American Mathematical Monthly*, v. 92, p. 3–23, 1985. Citado na página 46.

LARSON, S. M. et al. *Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology*. 2002. Citado na página 18.

MAJITHIA, S. et al. Triana: a graphical Web service composition and execution toolkit. In: *Proceedings. IEEE International Conference on Web Services, 2004.* IEEE, 2004. p. 514–521. ISBN 0-7695-2167-3. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1314777>. Citado na página 13.

MASTROIANNI, C. et al. A scalable super-peer approach for public scientific computation. *Future Generation Computer Systems*, v. 25, n. 3, p. 213–223, 2009. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X08001209>. Citado 4 vezes nas páginas 25, 29, 30 e 31.

MEDEIROS, J. W. et al. Scientific workflow systems. *NSF Workshop on Workflow and Process Automation: State-of-the-art and Future Directions*, 1996. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.6464>. Citado 2 vezes nas páginas 13 e 17.

MURATA, Y. et al. Implementation and evaluation of a distributed and cooperative load-balancing mechanism for dependable volunteer computing. In: *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 2008. p. 316–325. ISBN 978-1-4244-2397-2. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4630100>. Citado 2 vezes nas páginas 25 e 29.

MUTKA, M. W.; LIVNY, M. The available capacity of a privately owned workstation environment. *Performance Evaluation*, v. 12, n. 4, p. 269–284, 1991. Disponível em: <http://www.sciencedirect.com/science/article/pii/016653169190005N>. Citado na página 13.

PELTZ, C. Web services orchestration and choreography. *Computer*, v. 36, n. 10, p. 46–52, Oct 2003. ISSN 0018-9162. Citado na página 13.

RIUS, J. et al. Incentive mechanism for scheduling jobs in a peer-to-peer computing system. *Simulation Modelling Practice and Theory*, v. 25, p. 36–55, 2012. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1569190X12000305>. Citado 4 vezes nas páginas 25, 26, 27 e 29.

SCIENCETALK e. *Desktop grids: Connecting everyone to science*. 2011. Citado na página 19.

TAYLOR, I. J. et al. Scientific versus Business Workflows. In: *Workflows for e-Science: Scientific Workflows for Grids*. [S.l.: s.n.], 2007. cap. 2, p. 9–16. Citado na página 17.

Wen Dou et al. A P2P approach for global computing. In: *Proceedings International Parallel and Distributed Processing Symposium*. IEEE Comput. Soc, 2003. p. 6. ISBN 0-7695-1926-1. ISSN 1530-2075. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1213451>. Citado 4 vezes nas páginas 25, 26, 29 e 30.

ZHAO, Z.; YANG, F.; XU, Y. PPVC: A P2P volunteer computing system. In: *2009 2nd IEEE International Conference on Computer Science and Information Technology*. IEEE, 2009. p. 51–55. ISBN 978-1-4244-4519-6. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5234999>. Citado 6 vezes nas páginas 13, 25, 26, 29, 30 e 31.

ZHOU, D.; LO, V. WaveGrid: a scalable fast-turnaround heterogeneous peer-based desktop grid system. In: *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006. p. 10 pp. ISBN 1-4244-0054-6. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1639267>. Citado na página 13.

ZUNIGA, J. C. et al. A loosely coupled architecture for automatic composition of web services applications. *Int. J. Metadata Semant. Ontologies*, v. 9, n. 3, p. 241–251, jul. 2014. ISSN 1744-2621. Citado 2 vezes nas páginas 15 e 22.