Programa Interdisciplinar de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Alexandre da Silva Veith

BSPonP2P: Modelo para Exploração da Computação Colaborativa em Aplicações BSP para Ambientes Grades P2P

São Leopoldo, 2014

UNIVERSIDADE DO VALE DO RIO DOS SINOS — UNISINOS
UNIDADE ACADÊMICA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA
NÍVEL MESTRADO

ALEXANDRE DA SILVA VEITH

BSPONP2P: MODELO PARA EXPLORAÇÃO DA COMPUTAÇÃO COLABORATIVA EM APLICAÇÕES BSP PARA AMBIENTES GRADES P2P

São Leopoldo
2014

Alexandre da Silva Veith

BSPONP2P: MODELO PARA EXPLORAÇÃO DA COMPUTAÇÃO COLABORATIVA EM APLICAÇÕES BSP PARA AMBIENTES GRADES P2P

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos — UNISINOS

Orientador:
Prof. Dr. Rodrigo da Rosa Righi

São Leopoldo
2014

V431b Veith, Alexandre da Silva.
BSPonP2P : modelo para exploração da computação colaborativa em aplicações BSP para ambientes grades P2P, São Leopoldo - RS / por Alexandre da Silva Veith. – 2014.
95 f. : il. ; 30 cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, RS, 2014.
"Orientação: Prof. Dr. Rodrigo da Rosa Righi".

1. Sistemas de computação em grade. 2. Sistemas operacionais distribuídos (Computadores) I. Título.

CDU: 004.75

Catalogação na Publicação:
Bibliotecário Thiago Lopes da Silva Wyse - CRB 10/2065

UNISINOS

UNIVERSIDADE DO VALE DO RIO DOS SINOS
Ciências Exatas e Tecnológicas
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada – PIPCA

Alexandre da Silva Veith

BSPonP2P: Modelo para Exploração da Computação Colaborativa em Aplicações BSP para Ambientes Grades P2P.
BSP através replicação e migração para aumento de desempenho

Dissertação apresentada à Universidade do Vale do Rio dos Sinos – Unisinos, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Aprovado em 29/08/2014

BANCA EXAMINADORA

Antonio Marcos Alberti – INATEL

Rodrigo da Rosa Righi – Unisinos

Cristiano André da Costa – Unisinos

Prof. Dr. Rodrigo da Rosa Righi (Orientador)

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Cristiano André da Costa
Coordenador PPG em Computação Aplicada

Aos meus pais, amigos, filho e eposa.

If I have seen farther than others,
it is because I stood on the shoulders of giants.
SIR ISAAC NEWTON

AGRADECIMENTOS

Primeiramente a Deus pelas oportunidades;

À minha familia por me inspirar a acreditar nos meus objetivos;

À Unisinos por oferecer meios para que o trabalho em questão fosse desenvolvido;

A todos os professores do PIPCA pelo relevante trabalho que realizam;

Especialmente ao meu orientador, Dr. Rodrigo da Rosa Righi, por acreditar e conduzir os trabalhos rumo aos resultados.

"*Ninguém abre um livro sem que aprenda alguma coisa*".
(Anônimo)
"*Dá-me sabedoria e conhecimento, pois confio nos teus mandamentos*".
(Salmos 199,66).

RESUMO

Tecnologias constantemente estão avançando nas áreas de sistemas distribuídos e computação paralela. Isso acontece porque a compra de equipamentos eletrônicos está acessível, por isso empresas, cada vez mais, estão apostando em soluções baratas para que todos tenham acesso. Consequentemente, existe um problema que é o desperdício na utilização destes equipamentos. Grande parte de seu tempo esses equipamentos ficam ociosos. Nesse contexto, esta dissertação apresenta o modelo BSPonP2P para minimizar o problema, buscando aproveitar esse desperdício para algum fim útil. O BSPonP2P utiliza uma abordagem de Computação em Grade P2P, ela visa utilizar os equipamentos para execução de computação útil. O objetivo é fazer as execuções de maneira concorrente. O BSPonP2P cria um ambiente com abordagens baseadas nos modelos estruturado e não estruturado vindos da arquitetura P2P, o que foi implementado para agilizar o gerenciamento da comunicação e das informações dentro da rede. Outro diferencial do modelo proposto é a utilização do modelo de programação paralela *Bulk Synchronous Parallel* (BSP), que cria um ambiente para execução de processos, validando as dependências e aprimorando a comunicação entre eles. A partir de avaliações de métricas como memória, computação, comunicação e dados de equipamentos, é criado um índice denominado de PM. Esse índice é avaliado periodicamente para definir migrações conforme a variável de ambiente α, que está diretamente ligada às barreiras das *supersteps*. A partir de avaliações obtidas nas distribuições de processos em um ambiente heterogêneo criado, o modelo BSPonP2P se mostrou eficaz porque ele obteve bons resultados, como, por exemplo, na simples execução da aplicação, comparando com a execução do BSPonP2P, houve um aumento menor que 4% no tempo de execução. Além disso, na execução de 26 processos com 2000 *Supersteps* e $\alpha = 16$, obteve-se um ganho de 6% a partir de 24 migrações de processos. Sendo assim, como contribuição científica, optou-se pela utilização de redes em Grades P2P com aplicações BSP, usando métricas como memória, computação, comunicação e dados de equipamentos para avaliação do ambiente. Além de, serviços como migração e *checkpoint* que possibilitaram um bom desempenho do modelo.

Palavras-chave: Computação em Grade P2P. Migração. Sistemas Distribuídos. BSP. *Checkpoint*.

ABSTRACT

Technologies are constantly advancing in the areas of distributed systems and parallel computing. This is because the purchase of electronic equipment is accessible, so companies increasingly are betting on cheap solutions for everyone to access. Accordingly, there is a problem that the wasteful use of such equipment. Most of these have access to the execution of computation, however, a large part of their time sit idle. In this context, this dissertation proposal presents BSPonP2P model to minimize the problem trying to enjoy this waste for any useful purpose. In the proposed model, a P2P Desktop Grid that seeks to use equipment to perform useful computing competitor among its users and Desktop Grid network users way approach will be used. The BSPonP2P will create an environment with models based on structured and unstructured P2P architecture approaches coming, that will be implemented to streamline the management and communication of information within the network. Another difference that the proposed model will have is the use of Bulk Synchronous Parallel (BSP) parallel programming model, which creates an environment for process execution dependencies validating and improving the communication between them. From reviews of metrics such as memory, computation, data communications equipment and an index called PM is created. This index is periodically evaluated to define migration as the environment variable α, which is directly linked to the supersteps' barriers. Based on the ratings obtained from the distributions of processes in a heterogeneous environment created, BSPonP2P model is demonstrated effective. This is because the model had good results, for example, the simple execution application compared to running the BSPonP2P there was a smaller increase than 4% in execution time. Moreover, the implementation of 26 cases with 2000 *supersteps* and $alpha = 16$ yielded a gain of 6% from 24 migration process. Thus, atom scientific contribution opted for the use of networks in P2P Grids with BSP applications using metrics such as memory, computation, communication and data equipment for environmental assessment.

Keywords: P2P Desktop Grid. Migration. Distributed System. BSP. Checkpoint.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Na situação econômica em que se vive atualmente, existem inúmeras preocupações, que envolvem ambientes domésticos, empresariais e acadêmicos, tais como: (i) custos de investimentos, orçamentos limitados; (ii) desempenho, sendo ágil e eficiente e utilizando o máximo do que está disponível; (iii) restrições físicas, envolvendo dimensão e capacidade dos recursos oferecidos; (iv) meio ambiente, buscando a redução da emissão de gases e consumo de energia.

A fim de encontrar soluções para essas preocupações, paradigmas, através dos anos, vêm sendo quebrados e novos conceitos estão surgindo. Conceitos como a computação de alto desempenho e a utilização de sistemas distribuídos podem ser citados porque utilizam as perdas computacionais para a computação útil (MOHAMED; AL-JAROODI; JIANG, 2014). Esses conceitos baseiam-se na estruturação dos algoritmos para que possam ser executados paralelamente e de maneira distribuída. Isso significa que as execuções podem ocorrer ao mesmo tempo em equipamentos diferentes, não necessitando estarem eles no mesmo local e bastando estarem conectados.

A conexão desses equipamentos permite criar ambientes de partilha de computação e as seguintes tecnologias podem ser citadas: (i) Grades Computacionais (QURESHI et al., 2014), que utilizam os recursos computacionais de ambientes empresariais ou acadêmicos para a execução de computação. Nesse ambiente, geralmente os recursos computacionais são dedicados e o ambiente é estático. Além disso, a rede é controlada por pessoas que possuem conhecimento técnico da área; (ii) *Peer-to-Peer*(P2P)(ANDROUTSELLIS-THEOTOKIS; SPINELLIS, 2004), que utiliza os recursos computacionais (computadores pessoais, *notebooks*, *tablets* e *smartphones*) de ambientes domésticos. O foco principal desta tecnologia é o compartilhamento de arquivos. Além disso, o ambiente é dinâmico e os recursos não são dedicados.

Através da necessidade computacional e a presença da ociosidade computacional em usuários finais, a criação de meios para a sua utilização traz ganhos, como: (i) Econômicos (CASTRO et al., 2013), aprimorar a utilização dos recursos computacionais ao invés de investir em novos equipamentos; (ii) Ecológicos (NESMACHNOW et al., 2013), menor consumo de energia e emissão de gases; e (iii) Desempenho (LAREDO et al., 2014), maior velocidade nas execuções devido à quantidade de recursos disponíveis.

Um meio que viabiliza a utilização da ociosidade em ambientes domésticos é a arquitetura *overlay* (rede virtual) P2P (SHAH; KIM, 2014). Isso porque ela faz boa gestão de ambientes dinâmicos e heterogêneos, e ainda este tipo de tecnologia controla, de maneira eficiente, a entrada e saída de equipamentos da rede. Outras características que podem ser citadas são: praticidade no controle de *firewalls* e DNS e ter protocolos específicos para a gestão de redes de alta escala. Porém, o foco principal deste tipo de tecnologia é o compartilhamento de arquivos e não de computação. Ao contrário da rede P2P, a Computação em Grade tem como foco principal o compartilhamento para computação. Dessa forma, ao se utilizarem os pontos positivos das Grades Computacionais (*Desktop Grid*) e P2P, surge outra estratégia denominada de Computação

em Grade P2P (P2P *Desktop Grid*) (HALIM; ABIDIN; LATIP, 2012).

No entanto, a utilização de ambientes em Grades P2P para computação, como em qualquer arquitetura distribuída, necessita de gerenciamento dos recursos computacionais, processos e comunicação. Esse gerenciamento é feito pelo controle e balanceamento de carga de forma que a utilização dos recursos gere bom desempenho tanto em relação à velocidade de execução quanto ao aproveitamento da ociosidade. Entretanto, não existe a melhor forma de balaceamento e controle, e, sendo assim, surgem diversas abordagens, por exemplo: (i) Lin et al. (2008), avalia o impacto da migração de processos; (ii) Camargo, Castor e Kon (2010), estuda a aplicação de replicação para segurança; e (iii) Godfrey et al. (2004), aborda a criação de diretórios para controle.

Para atingir o balanceamento de carga, juntamente com o gerenciamento dos recursos, é necessário dividir os trabalhos em pequenos blocos (processos) e conseguir executá-los paralelamente. Uma ferramenta que auxilia isso é a programação paralela. Na programação paralela existem diversas técnicas, como: (i) Mestre/Escravo (HUANG et al., 2008); (ii) *Bag of Tasks* (CELAYA; MARCHAL, 2010); e (iii) *Bulk-Synchronous Parallel* (VALIANT, 1990). O modelo BSP destaca-se na execução de trabalhos que possuam dependência entre os processos. Valiant (1990) montou este tipo de modelo para estruturação, análise e programação de sistemas paralelos, em que as aplicações são compostas por conjuntos de processos que são executados em *supersteps*. Cada um deles é subdividido em três fases: (i) computações locais em cada processo; (ii) utilização de mensagens para comunicação global; e (iii) barreira de sincronização. Conforme Righi (2012), o BSP é um modelo usado em aplicações paralelas de sucesso, principalmente em aplicações de alto desempenho. Isso ocorre por possuir uma organização simples e ser possível tirar uma função de custo da aplicação sob uma determinada arquitetura de máquina. Além disso, o modelo BSP é aplicável a diversos tipos de redes, como por exemplo: (i) Miriam e Easwarakumar (2011) e (ii) Kondo et al. (2004), Computação em Grade de *Desktop*; (iii) Basu et al. (2013), P2P; e (iv) Vaquero et al. (2008), Computação em Nuvem.

Sendo assim, pela combinação da arquitetura de Grade P2P com aplicações BSP é possível criar um ambiente eficiente de partilha de computação. A eficiência é dada pela utilização dos equipamentos ociosos, bem como pela diminuição do tempo total de execução. Isso porque o tipo de arquitetura permite a gestão e o controle da dinamicidade e heterogeinidade do ambiente. O BSP cria um ambiente para execução de trabalhos que possuam tarefas com dependências, permitindo a comunicação entre elas. Além disso, permite a gestão do escalonamento e reescalonamento dos processos.

1.1 Problema *(Problem Statement)*

Com base no contexto explicado acima, esta dissertação possui a seguinte sentença problema: *Como explorar computação colaborativa em Grades P2P para execução de aplicações BSP com eficiência?*

A eficiência é dada a partir da diminuição do tempo total de execução e a utilização de equipamentos ociosos. Consequentemente, trazer um benefício ecônomico, não necessitando investimento para a compra de equipamentos de capacidade computacional. O ambiente aproveitará os recursos computacionais de forma que eles estejam disponíveis e não haja grande custo para alocação do processo. As métricas para o cálculo desse custo são as chaves para a definição da eficiência da exploração da computação colaborativa.

1.2 Objetivos

O objetivo principal é: *criar um modelo para computação colaborativa onde será aproveitada a ociosidade dos recursos computacionais para execução de aplicações BSP.* E neste novo ambiente, busca-se aprimorar as execuções através de avaliações dos custos de comunicação e de computação. Para isso, os seguintes pontos devem ser avaliados como objetivos secundários:

- Criar uma heurística eficiente de balanceamento de carga;
- Reduzir o tempo de execução dos processos;
- Utilizar uma rede que permita controlar as instabilidades (entradas, saídas e demandas do usuário);
- Implementar um *middleware* que abstraia as primitivas da implementação e dimensione a execução de processos;
- Analisar o desempenho da execução pelos resultados obtidos.

1.3 Organização do Texto

O restante desta dissertação é organizada da seguinte maneira: O Capítulo 2 apresenta a fundamentação teórica dos assuntos: (i) Arquiteturas de sistemas distribuídos, onde são evidenciadas as características das arquiteturas *Peer-to-Peer*, Computação em Grade e Grades P2P; (ii) Programação paralela, que detalha BSP e balanceamento de carga; e (iii) Simuladores, em que se demonstram alternativas de simuladores (PeerSim e SimGrid). O Capítulo 3 demonstra o contexto atual de trabalhos que seguem a linha do modelo proposto e ao final os compara. O Capítulo 4 expõe o modelo BSPonP2P através de decisões de projeto, taxonomia, modelo de aplicação e serviços oferecidos. O Capítulo 5 apresenta as técnicas e ferramentas utilizadas para implementação. O Capítulo 6 avalia os resultados obtidos. E, por fim, o Capítulo 7 conclui e propóe os trabalhos futuros.

2 FUNDAMETAÇÃO TEÓRICA

Este capítulo tem um caráter explicativo, apresentando alguns conceitos e terminologias que serão utilizados nos próximos capítulos. Mostra-se a relevância de simulações e do balanceamento de carga em áreas de processamento paralelo e distribuído. Também se descreve o modelo de execução BSP, que é empregado no modelo de aplicação envolvido com o proposto para o processo de reescalonamento. Além disso, são apresentadas as características de ambientes distribuídos para conduzir ao ambiente em que cada uma pode ser encaixada.

Este capítulo está segmentado em três seções: Seção 2.1 - demonstra os pontos fortes e fracos da P2P, Computação em Grade e Grades P2P, bem como as suas características e aplicações; Seção 2.2 - apresenta o modelo BSP e pontos referentes ao balanceamento de carga; e Seção 2.3 - oferece alguns conceitos de simulação, bem como ferramentas para execução de simulação.

2.1 Arquitetura de Sistemas Distribuídos

A criação de meios de compartilhamento computacional é uma ferramenta que auxilia no aprimoramento da utilização dos recursos computacionais. Isso acontece devido à viabilidade de submissão de tarefas, utilização de mémória e troca de arquivos entre recursos computacionais. Com isso, é ampliada a capacidade computacional, o que auxilia na solução de problemas que exijam capacidade computacional. A construção deste ambiente geralmente é resumida na criação de uma *overlay* (TRAVERSO et al., 2014) sobre a rede física (*Internet*).

Para melhor entender o funcionamento na criação destes meios, são explicadas algumas estruturas, como: *Peer-to-Peer*, características entre redes estruturadas e não estruturadas, bem como algumas ferramentas para aplicação desta arquitetura; Computação em Grade, apresentando conceitos e caracterísitcas; e Grades P2P, o que resulta entre a combinação entre Computação em Grade e *Peer-to-Peer*.

2.1.1 *Peer-to-Peer* (P2P)

Coulouris, Dollimore e Kindberg (2007) acreditam que o modelo *Peer-to-Peer* surgiu em consequência do crescimento da *Internet*. Neste modelo, muitos *hosts* colaboram entre si de maneira uniforme, tanto em compartilhamento de arquivos, quanto na distribuição de informações, compartilhando a execução de tarefas de aplicativos e outros serviços que exploram sua maior eficácia. Assim, torna a rede um grande centro de processamento e de integração.

De acordo com AlTuhafi, Ramadass e Chong (2013), *Peer-to-peer* tem sido amplamente utilizado atualmente em redes. A limitação que sistemas cliente/servidor têm em relação a número de clientes que podem ser servidos leva à criação de sistemas como P2P. Isso acontece a fim de possibilitar a partilha de conteúdos ou de recursos computacionais entre os participantes da rede.

A utilização de uma rede P2P ((JAISWAL et al., 2013), (PING et al., 2004) e (MELAB; MEZMAZ; TALBI, 2005)) é algo poderoso para a solução de problemas complexos, pois, neste tipo de topologia, os recursos são compartilhados por uma rede *overlay*. Neste ambiente, os usuários que desejam aperfeiçoar a utilização de seus recursos computacionais (domésticos ou empresariais) podem compartilhá-los através desta rede virtual. Pois, na maioria das vezes, os equipamentos são destinados para o acesso à *Internet* (redes sociais, pesquisas), *downloads* e ferramentas de manuseio de texto. Nesses tipos de utilização, não se aproveita a capacidade computacional disponível, não se exige empenho do processador, acarretando vários momentos de ociosidade.

De acordo com Ogata et al. (2010a), a rede P2P possui algumas facilidades em lidar com *firewalls*, NATs e outros dispositivos de segurança. Além desses benefícios, a rede também trabalha de uma forma descentralizada, o que traz maior segurança ao ambiente, não necessitando que haja uma centralização na tomada de decisão.

Para demonstrar a importância da implementação deste tipo de ambiente, seguem alguns exemplos de tecnologias criadas para facilitar a implementação de redes P2P:

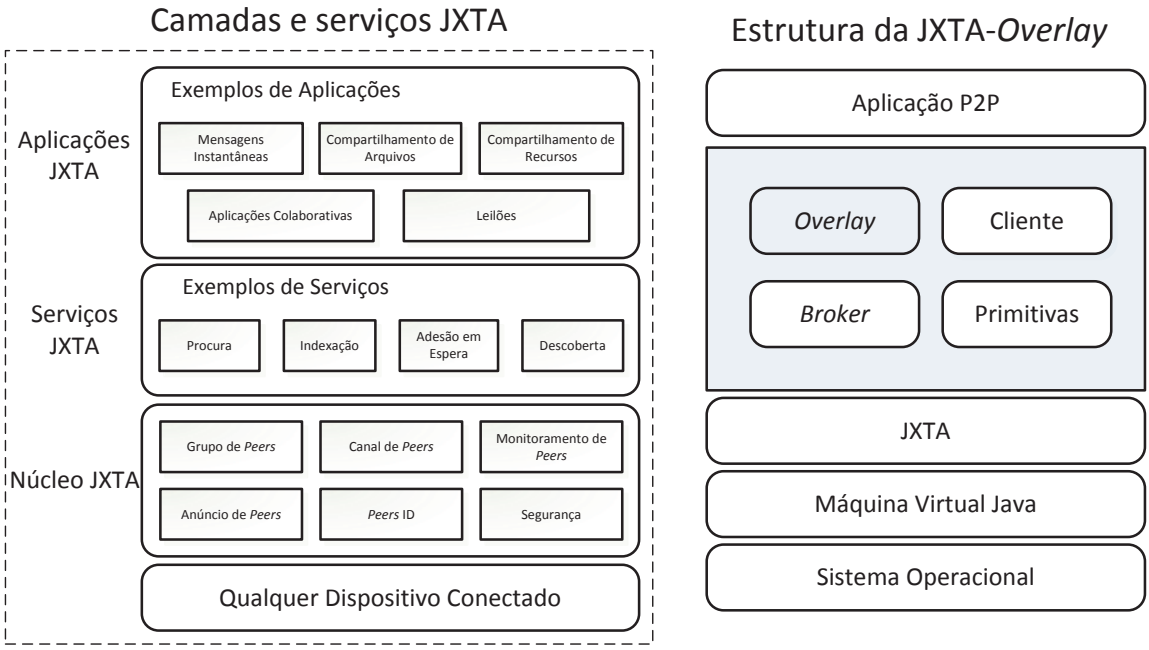
- **BOINC**: Para disseminar o conceito de utilização de rede P2P no aproveitamento de recursos, várias pesquisas e ferramentas estão sendo geradas para abstrair e criar padronizações, que, atualmente, não existem para redes P2P. Dessa forma, vários conceitos são aplicados para estruturar a rede. Por exemplo, o projeto Anderson (2004) apresenta a estrutura BOINC, que abstrai a implementação da arquitetura, gerando, assim, uma facilidade na criação de redes distribuídas. Muitos projetos são criados, utilizando esta abordagem a fim de solucionar problemas complexos, como, por exemplo: Cerin e Takoudjou (2013), Anderson et al. (2002), Pande et al. (2003) e Stainforth et al. (2002).

- **JXTA**: Este *middleware*, conforme Matsuo et al. (2009), Zhang et al. (2014), Ogata et al. (2010b) e Kolici et al. (2011), é um agrupamento de protocolos que permite diferentes dispositivos se comunicarem e colaborarem entre si. JXTA oferece uma plataforma para cobertura básica necessária para desenvolver redes P2P. Com a utilização desta ferramenta, é possível conectar equipamentos em redes privadas na *Internet*, passando pelos *firewalls* existentes. O projeto JXTA-*Overlay* (UMEZAKI et al., 2013) é um empenho para utilizar a tecnologia JXTA para construção de uma *overlay* no topo de uma JXTA oferecendo um conjunto básico de primitivas que são comuns no desenvolvimento de aplicações baseadas em JXTA. As primitivas, geralmente, são oferecidas por este tipo de arquitetura: descoberta de equipamentos e recursos; alocação de recursos; submissão e execução de tarefas; compartilhamento, descoberta e transmissão de arquivos e dados; comunicação instantânea; funcionalidade para grupo de equipamentos (grupos, salas, etc.); e monitoramento de equipamentos, nós, tarefas e grupos.

  A arquitetura interna da JXTA-*Overlay* é estruturada conforme os blocos destacados na Figura 1, permitindo utilizá-los para o desenvolvimento da rede virtual, facilitando a cri-

ação de redes P2P estruturadas e não estruturadas, ampliando conexões com dispositivos como PDAs, telefones inteligentes, computadores, *notebooks*, entre outros.

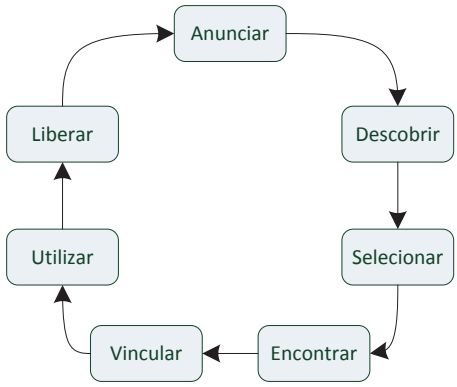
Figura 1 – Estrutura, camadas e seviços da JXTA-*Overlay*



Fonte: Kolici et al. (2011)

Para a compreensão do funcionamento da rede P2P é necessário o entendimento em relação à colaboração de recursos. Conforme Bandara e Jayasumana (2013) foram identificadas sete fases na parte de colaboração em P2P (Figura 2). Essas fases estão identificadas por causa do entendimento conceitual. Uma implementação real pode agrupar várias fases em uma única. Sistemas específicos podem retirar algumas delas, dependendo dos requisitos da aplicação. No entanto, há complexidades, detalhes de implementação e trabalhos de relatos disponíveis em cada uma das fases. As fases são as seguintes: Anunciar - cada equipamento anuncia seus recursos e sua capacidade, utilizando uma ou mais especificações de recursos; Descobrir - equipamentos devem enviar mensagens de exploração para, proativamente, descobrir e moldar um repositório local de especificações de recursos úteis; Selecionar - selecionar um grupo de recursos que satisfaçam os requerimentos dados pela aplicação; Encontrar - onde, a partir de uma consulta, os equipamentos atendam parcialmente ou completamente às especificações dos recursos que sejam apropriados ou capazes de trabalhar em conjunto; Vincular - uma vez que um subconjunto de recursos seja encontrado e que atenda aos requerimentos da aplicação, é necessário garantir que eles estejam disponíveis para uso; Utilizar - utilizar o melhor subconjunto de recursos encontrados que satisfaçam os requerimentos da aplicação e restrições para a execução das tarefas da aplicação para os recursos adquiridos; Liberar - liberar recursos quando a demanda de aplicação diminui, ou a tarefa está completa, ou a ligação expira.

Figura 2 – Fases na colaboração de recursos



Fonte: Bandara e Jayasumana (2013)

Arquiteturas P2P, de maneira geral, podem ser categorizadas como *overlays* estruturadas e não estruturadas (LIU et al., 2010). As características de cada uma são comparadas na Tabela 1. *Overlay* não estruturada é atrativa, pois sua construção e futuras manutenções são relativamente simples. Além do mais, elas distribuem as informações dos recursos entre os nós no sistema enquanto proveem de elasticidade e balanceamento de carga. P2P estruturadas são conhecidas pela alta escalabilidade e algumas garantias de desempenho e, assim, são utilizadas em alta escala e em ambientes relativamente robustos. Estes sistemas usam Tabelas Distribuídas *Hash* (DHT - *Distributed Hash Table*) (YING; JIANG, 2010) para indexar os recursos. Cada nó possui uma DHT e um recurso tem uma identificação única chamada de chave. Uma rede *overlay* determinística é utilizada para roteamento de mensagens que anunciam especificações e consultas de recursos. Chord (WOUNGANG et al., 2014) e CAN (RATNASAMY et al., 2001) são algumas soluções conhecidas para fazer *overlays* estruturadas.

A seguir, serão explicadas algumas soluções de arquiteturas de *overlays* não estruturadas:

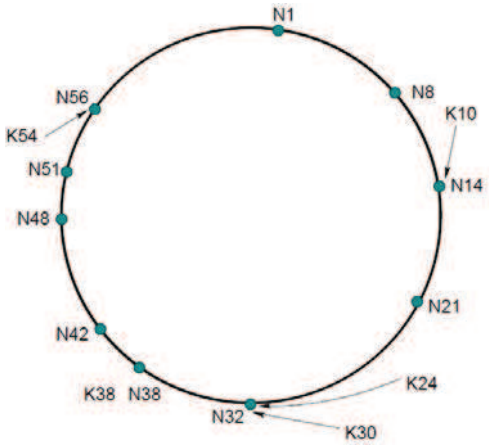
- Chord: O protocolo Chord (STOICA et al., 2003) é utilizado na solução de um problema frequente em aplicações P2P, que é estabelecer em qual equipamento está armazenado determinado valor. Este protocolo gera a solução de uma maneira descentralizada, distribuindo rapidamente a computação da função *hash*, mapeando chaves para nós responsáveis por elas. As chaves são criadas para nós com *consistent hashing*, que em resumo, criam para cada nó e chave uma identificação *m-bit* (YOON; CHO; LEE, 2000) usando SHA-1 (WANG; YIN; YU, 2005) como função base de *hash*. Conforme pode ser visto na Figura 3, as identificações são ordenadas em um "círculo de identificação", chamado de *Chord ring*. Este método é desenvolvido para permitir a entrada e saída de nós da rede com a menor ruptura, além de, com alta probabilidade, a função de *hash* balanceia a rede (todos os nós recebem aproximadamente o mesmo número de chaves). Também com alta probabilidade, quando um certo equipamento N entra na rede (ou a deixa), somente ($\theta\frac{1}{N}$) frações de chaves são movidas para diferentes locais.

Tabela 1 – Comparação entre o modelo de sistema P2P estruturado e estruturado

	P2P não estruturado	**P2P estruturado**
Construção *overlay*	Alta flexibilidade	Baixa flexibilidade
Recursos	Indexados localmente(tipicamente)	Indexados remotamente em uma tabela distribuída *Hash*
Consulta de mensagens	*Broadcast*(LYKOURGIOTIS et al., 2014) ou *Random Walk* (LI et al., 2014)	*Unicast* (IORGA et al., 2014)
Localização de conteúdo	Melhor esforço	Garantido
Desempenho	Não previsível	Previsão amarrada
Tipos de objetos	Mutável com muitos atributos complexos	Imutável, com simples atributos
Overhead da manutenção da *overlay*	Relativamente baixa	Moderado
Grande volume de entradas na rede e falhas	Suporta altos índices de falhas	Suporta moderados índices de falhas
Ambientes de aplicação	Pequena escala ou ambientes altamente dinâmicos com objetos (i)mutáveis, P2P móveis	Alta escala e ambientes relativamente estáveis com objetos imutáveis, compartilhamento *Desktop* de arquivos
Exemplos	Gnutella (RIPEANU, 2001), LimeWire (BRETZKE; VASSILEVA, 2003), Kazaa (GOOD; KREKELBERG, 2003)	Chord, CAN, Pastry (ROWSTRON; DRUSCHEL, 2001), Kademlia (MAYMOUNKOV; MAZIèRES, 2002)

Fonte: Bandara e Jayasumana (2013)

Figura 3 – Círculo de 10 nós armazenando cinco chaves



Fonte: Stoica et al. (2003)

Sendo assim, pelo protocolo Chord, é oferecida uma ferramenta poderosa de primitivas: por meio de uma chave, é possível determinar o nó responsável por armazenar o valor dessa, e isso é feito com muita eficiência. Para isso, cada nó em uma rede de N-equipamentos faz somente $(\theta(\log N))$ manutenções na tabela de roteamento dos outros nós e são resolvidas todas as procuras via $(\theta(\log N))$ mensagens para outros equipamentos. Além disso, oferece simplicidade, provável correção, desempenho, mesmo em cenários com concorrência de entrada e partida de nós.

- *Content addressable network* (CAN): Ratnasamy et al. (2001) diz que um sistema P2P requer, pelo menos, um mecanismo de fácil indexação para localização. Estes sistemas de indexação são chamados de CAN. A aplicabilidade de CANs não se limita apenas a sistemas *Peer-to-Peer*, também podem ser utilizados para gerenciamento e armazenamento em alto escala, como em OceanStore, Farsite e Publius. Todos estes sistemas requerem uma inserção e recuperação de conteúdo largamente distribuídas, e um mecanismo de indexação é um componente essencial nestas estruturas. Outra aplicação potencial de CAN é na construção de serviços de resolução de autoridades em *wide-area* (parecido com DNS).

2.1.2 Computação em Grade

Ernemann et al. (2002) afirmam que Computação em Grade é a forma de oferecer acesso direto a recursos independentes, escassos e limitados, fazendo com que cooperem entre si em suas devidas redes. Desse modo, é possível obter maior desempenho em vez de desperdiçá-los no momento em que ficam ociosos.

Recursos geralmente são escassos em empresas, devendo-se fazer muito com pouco. Para tal, a imaginação deve fluir, buscando encontrar o melhor aproveitamento dos recursos existentes e gerando maiores resultados com menor investimento. Isso faz com que não seja necessário ficar constantemente trocando equipamentos, trazendo redução de custos e um possível investimento na equipe ou na ampliação dos recursos da empresa.

Conforme Yu e Buyya (2005), ferramentas complexas têm sido aprimoradas e criadas a partir da Computação em Grade. Redes simples podem tornar-se centros de processamento, pois, com a capacidade de distribuição entre os equipamentos, crescem as oportunidades, gerando melhores resultados no que se necessita fazer.

Ernemann et al. (2002) destacam que a divisão dos nós por meio de uma rede de Computação em Grade permite o particionamento de grandes tarefas que exigem mais dos equipamentos em processamento paralelo. Porém, para isso, é necessário considerar os recursos envolvidos, como a latência da rede e a largura de banda, pois terão grande influência na comunicação entre os nós, questão de fundamental relevância em ambientes distribuídos.

2.1.3 Computação em Grades P2P

De acordo com Choi et al. (2006), Computação de Grades P2P visa aproveitar computadores *desktops* ociosos que estão na ponta da *Internet*. Computação em Grades de *Desktop* é um tipo de Computação em Grade, porém entre essas duas tecnologias existem distintas e severas diferenças em termos de tipos e características de recursos, além dos tipos de compartilhamento. Computação em Grades de *Desktops* recentemente obtive um crescimento rápido de interesse e atração por causa do sucesso de exemplos populares como: Seti@home Anderson et al. (2002)

e Folding@Home Pande et al. (2003).

Da mesma forma, Zhao, Liu e Li (2011) acreditam que nas últimas décadas houve uma evolução muito rápida em Computação em Grade, o que trouxe alto desempenho e rendimento em alta escala para aplicações científicas, de engenharia, de negócios e militares. No entanto, sistemas em grade estão sendo confrontados com um significativo volume de mudanças no incremento da escalabilidade e questões de complexidade. Por outro lado, sistemas *Peer-to-Peer* (P2P) foram concebidos e desenhados para escalas e usuários massivos. Por influência de tecnologias de Computação em Grade com padrões e arquiteturas abertas, além de uma escalabilidade elegante e flexível das tecnologias P2P, a convergência das tecnologias Grade e P2P é eminente, particularmente em ambientes de Computação em Grade *Desktop*.

De acordo com Tomas, Caminero e Carrion (2013), um motivo pelo qual esta tecnologia vem crescendo é relacionado a questões ambientais, e tal preocupação no meio computacional pode ser denominada Computação Verde. Uma das questões que são levantadas nessas preocupações é o consumo de energia devido às limitações do tempo de vida das baterias, tornando crítico o gerenciamento de energia.

Conforme Halim, Abidin e Latip (2012), basicamente máquinas em Grades de *Desktop* são caracterizadas em **Cliente** (máquinas a usuários submetem grades de tarefas para execução), e **Trabalhador** (recursos computacionais nos quais as grades de tarefas podem ser executadas). No ambiente Grade P2P, no entanto, cada nó atua como um *peer*, que significa que o nó pode atuar como um trabalhador ou como um cliente. Geralmente, integrando as técnicas P2P em sistemas de Grade P2P, há severas vantagens em vários aspectos. Primeiramente, a integração permite modelos mais sofisticados de paralelismo, pelo que o P2P é capaz expandir o cenário para uma extensão maior. Ao contrário, Grades de *Desktop* típicas somente suportam aplicações paralelas relativamente simples. Em segundo lugar, P2P introduz mais heterogeinidade, além de estender as comunidades de usuários. Finalmente, ainda importante, P2P adiciona mais elastidade a sistemas de Grades de *Desktop* em termos de escalonamento e gerenciamento de tarefas.

De acordo com as principais diferenças citadas na Tabela 2, o aplicativo proposto por este estudo deve ser classificado no modelo Peer-to-Peer, pois qualquer computador poderá entrar ou sair da rede sem haver a necessidade de uma rede especial para isso, fazendo uso da rede TCP já utilizada na empresa. Outro ponto relevante é que o ambiente será dinâmico, em que vários nós entram e saem conforme a intenção do usuário em frente ao seu controle.

Tabela 2 – Diferenças entre os modelos

	Computação em Grade	**P2P**
Usuários	Multiprocessadores e *clusters*	*Desktops*
Redes	Rede de alta velocidade dedicada à Internet	Conexões comuns, geralmente TCP
Administração	Centralizada ou hierárquica	Distribuída
Aplicação	Aplicações complexas, como simulações em alta escala, análise de dados	Compartilhamento de arquivos em tempo real, dados *streaming*
Escala	Conecta um número relativamente pequeno de sites	A conexão é possível a partir de qualquer *desktop*
Segurança	Serviços seguros para o envio de tarefas e realização interativa	Protocolos para compartilhamento de arquivo
Participação	Estático ou mudando lentamente conforme a participação de nós ao longo do tempo	Dinâmico, controle sobre as entradas e saídas
Confiança	Usuários confiáveis	Não confiável, os usuários são anônimos

Fonte: Trunfio et al. (2006)

Com a pesquisa de Kondo et al. (2004), é demonstrado que, por meio de vários estudos, foi verificado que há uma alta disponibilidade de ciclos de CPU para uma grande coleção de máquinas *desktop*. O resultado desses estudos enumera alguns itens para caracterizar a utilidade dos recursos de uma Grade *Desktop*. Primeiramente, o monitoramento de disponibilidade de CPU não pode ser um bom modelo do que um aplicativo pode receber devido a idiossincrasias do sistema operacional, como pode ser visto no número de instâncias. Em segundo lugar, nesses estudos, não são contabilizadas as atividades relacionadas aos periféricos (*mouse*, teclado, etc.), que, em muitos sistemas, causam a suspenção da aplicação Grade *Desktop*. Em terceiro lugar, nos mesmos estudos, não são contabilizadas sobrecarga, limitações e políticas de acessos do computador utilizado pela Grade *Desktop*. Em quarto lugar, não fica claro se esses resultados trazem informações suficientes para compreender o comportamento de falhas de uma tarefa em uma aplicação que está rodando na Grade *Desktop*.

2.2 Programação Paralela

Um programa é considerado programação paralela quando é visto como um conjunto de partes que podem ser resolvidas concorrentemente. Cada parte é igualmente constituída por uma série de instruções sequenciais, mas que, no seu conjunto, podem ser executadas simultaneamente em vários processadores (KUPER et al., 2014). Na maneira sequencial, não há problemas com sincronismos e até mesmo em estruturação do código para que este se divida e possa ser executado em equipamentos diferentes. A programação paralela pode ser utilizada de maneira síncrona em que as tarefas necessitam ter uma determinada sequência para execução

ou de maneira assíncrona em que não existem dependências para execuções. Entrando nesse tipo de tecnologia, há diversos modelos de programação paralela. A seguir, são expostos alguns exemplos: Mestre/Escravo, *Bulk Synchronous Parallel* e *Bag-of-Tasks*.

2.2.1 *Bulk Synchronous Parallel* (BSP)

BSP (YZELMAN et al., 2014) é definido como um conjunto de processadores com memória local, interconectada por um mecanismo capaz de comunicação ponto a ponto e um de sincronização. Um programa BSP consiste de um conjunto de processos, uma sequência de *supersteps*, e uma barreira de sincronização, que é delimitada em intervalos de tempo. Em uma *superstep*, cada processo executa computações locais, envia mensagens para outros processos e, logo após, indica, por meio da chamada de um método de sincronização, que está pronto para uma barreira de sincronização. Quando todos os processos invocarem o método de sincronização e todas as mensagens estiverem entregues, iniciar-se-á a próxima *superstep*. Então, todas as mensagens enviadas durante a *superstep* anterior poderão ser acessadas por todos os beneficiários.

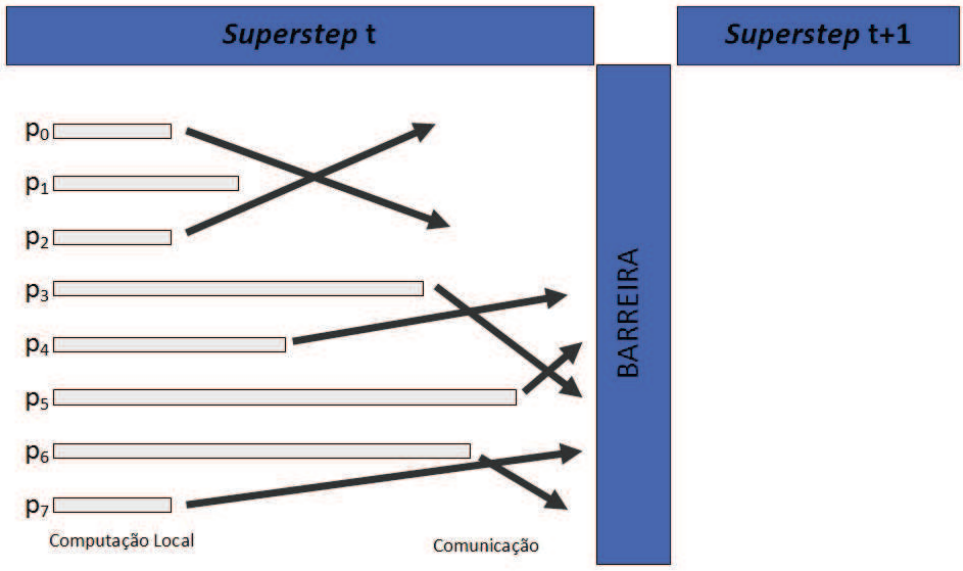
O modelo *Bulk Synchronous Parallel* (BSP), conforme Camargo, Castor e Kon (2010), é a divisão de programação em uma sequência de *supersteps* paralelos, em que cada *superstep* é dividido em computação e uma fase de comunicação (Figura 4), seguida de uma barreira de comunicação. Além disso, Righi (2012) complementa, informando que o BSP é muito parecido com programas sequenciais. Somente com o mínimo de informação extra necessária é fornecido o uso do paralelismo. Diferentemente de muitos sistemas de programação paralela, o BSP foi desenhado para ser uma arquitetura independente. Os programas podem ser executados sem mudança quando são movidos de uma arquitetura para outra. O tempo de execução do programa BSP pode ser computado a partir do seu texto e alguns parâmetros simples da arquitetura-alvo. O modelo BSP compreende uma coleção de processadores e cada um tem sua própria memória local e comunicação global.

Conforme Loulergue (2005), um grande número de arquiteturas pode ser visto como computação BSP. Por exemplo, máquinas com memória compartilhada podem ser utilizadas de maneira que cada processador acessa apenas uma pequena parte da memória compartilhada (que é, então, "privada") e comunicações podem ser executadas, utilizando uma parte dedicada da memória. Além disso, raramente a unidade de sincronização é um hardware, mas sim um software. O desempenho da computação BSP pode ser caracterizado em três parâmetros: o número de pares de processadores de memória; o tempo requerido para uma sincronização global; e o tempo para as entregas (fase de comunicação em que todos os processadores enviam e recebem mensagens).

2.2.2 Balanceamento de Carga

Jackson, Keleher e Sussman (2014), Shah, Veeravalli e Misra (2007) e Eager, Lazowska e

Figura 4 – Visão geral do funcionamento do BSP



Fonte: Elaborado pelo autor

Zahorjan (1985) acreditam que existem duas políticas básicas em algoritmos de balanceamento de carga - transferência e localização. A primeira decide se há a necessidade de iniciar o balanceamento de carga durante a execução do sistema. Por meio da utilização de informações de carga, é determinado quando o nó se torna elegível para atuar como *sender-initiated* (VYAS et al., 2014) (remetente, transferência de uma tarefa para outro nó) ou como *receiver-initiated* (NATARAJAN, 2014) (destinatário recebe uma tarefa de outro nó). A política de localização determina o processador adequado com sobrecarga. Dessa forma, são localizados equipamentos complementares donde pode ser recebida ou enviada carga para melhorar o desempenho geral do sistema. Políticas baseadas em localização podem ser amplamente classificadas em *sender-initiated*, *receiver-initiated* e *symmetrically-initiated* (IKEUCHI et al., 2014). Quando o balanceamento está sendo executado, informações são capturadas, tais como: número de tarefas esperando nas filas, taxa de chegada das tarefas, taxa de CPU para processamento, entre outras. Com base nas capturas das informações, é classificado em estático, dinâmico ou adaptativo.

Com a classificação criada por Casavant e Kuhl (1988), no período de execução, a topologia do sistema pode mudar, porém as características descritas da tarefa permanecem as mesmas. Assim, o escalonador pode gerar novas atribuições para os processadores das tarefas, servindo como escalonamento antes da mudança da topologia. Já em algoritmos dinâmicos, constantemente, há atualização das características das tarefas.

2.3 Simuladores

A criação de aplicações que utilizam sistemas distribuídos, na sua grande maioria, é complexa e confusa, pois trata com sistemas de escalas variadas, estruturas diferenciadas e objetivos diferentes (balanceamento de carga, roteamento, etc.). Como os resultados não são certos, o

tempo utilizado para a criação da aplicação e testes não podem ser grandes. Por isso, a utilização de simulação é um bom negócio, pois ela é uma ferramenta dinâmica que auxilia no desenvolvimento e nos testes, não necessitando da configuração e da instalação da aplicação em vários computadores. Além disso, em uma mesma plataforma, se consegue fazer tudo que se deseja: desenvolver, testar e colher resultados.

Conforme Basu et al. (2013), quando pesquisadores desenvolvem uma tecnologia como protocolos ou algoritmos de roteamento, é necessário testá-la para se assegurar de determinadas propriedades, como qualidade, robustez e sobrecarga. Para desenvolver aplicações reais que se tornam complexas devido à imprevisibilidade em relação ao número de nós, heterogeneidade de equipamentos e sua entrada e saída na rede, consequentemente, será necessário um gasto elevado em relação a testes de usuários e equipamentos. A partir dessas premissas, fica claro que a simulação é algo atrativo devido a sua maior facilidade de implementação e testes.

O objetivo da simulação é gastar mais tempo obtendo resultados do que corretamente projetando-o. O tempo gasto para obter os resultados de desempenho é algo mandatório. O tempo gasto para uma execução completa não pode ser alto. Por isso, conforme Ernst-Desmulier et al. (2006), em grande parte das vezes a simulação obtém melhores resultados do que a real execução.

De acordo com Gu, Tang e Chen (2009), os simuladores podem ser classificados em duas categorias com base em pacotes e nível de aplicação. Simuladores baseados em pacotes calculam o atraso, largura de banda e roteamento de cada pacote (gerado ou usado na simulação). Simuladores em nível de aplicação não contabilizam pacotes e são contabilizados largura de banda e atraso por meio dos pontos finais. Em simuladores em nível de aplicação, geralmente, são usados termos "baseado em fluxo"ou "baseado em mensagem"para descrever como é testada a comunicação entre os equipamentos.

Computação distribuída é algo vasto, com várias aplicações complexas, e as ideias são frequentes nesta área, porém, na maioria das vezes, complexas e de difícil implementação, porque os trabalhos com alta escala em relação à dimensão e aos algoritmos de balanceamento de carga são constantemente alterados. Sendo assim, foi necessária a criação de simuladores para facilitar aplicações desses ideais. Algumas ferramentas relacionadas a P2P são expostas a seguir:

- **PeerSim**: PeerSim (MONTRESOR; JELASITY, 2009) é um ambiente extremamente escalável que suporta cenários dinâmicos, tais como a rotatividade alta (*churn*) e outros modelos de falhas. Esta ferramenta foi desenhada em módulos para facilitar a configuração, sendo que o modelamento da rede é feito a partir de uma lista de nós, em que cada um possui uma lista de protocolos. O motor de simulação pode ser baseado em ciclos (protocolos são executados em alguma ordem especificada) ou em evento. Esses componentes podem ser completamente configurados e alterados. PeerSim provê de componentes simples com funcionalidades básicas, mas usuários possuem a permissão para trocá-los conforme suas implementações resultantes de suas preferências. Cada simulação é especificada por um simples texto de configuração similar ao arquivo de propriedade

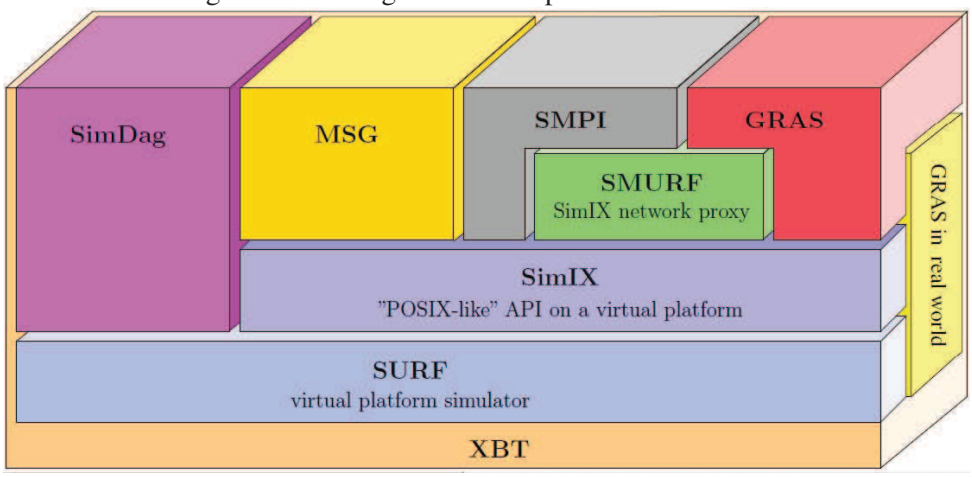
Java. As propriedades definem a implementação (Classes Java) dos componentes, e ele também especifica os parâmetros numéricos ou texto de cada um dos componentes.

Destaques dos recursos:

- Escalabilidade: Estruturas de dados internos têm uma pegada muito pequena de memória. Para dar alguns exemplos anedóticos: com protocolos simples e do motor baseado no ciclo, o limite de tamanho da rede é praticamente toda a memória disponível; redes de mais de 10^7 nós foram simulados na memória 4G;
- Modularidade: Todos os componentes do simulador, bem como os do observador e inicializador, podem ser livremente configurados. Além disso, a aplicação dos componentes do próprio simulador pode ser personalizada. É possível substituir os componentes-chave, tais como o nó de rede e a fila do motor baseado em eventos, simplesmente implementando a interface apropriada e adicionando uma linha ao arquivo de configuração;
- Abstração de grafos: Esta é uma característica forte de PeerSim. Pode tratar redes sobrepostas como grafos e fornecer vários inicializadores (modelos aleatórios e pequenos cenários, etc.), bem como os observadores, incluindo diâmetro da rede, *clustering* e conectividade. Grafos de rede de sobreposição podem ser exportados em vários formatos populares para o desenho e análise;
- Abstração de vetores: Módulos PeerSim existentes permitem que os desenvolvedores enriqueçam suas simulações, bastando escrever algumas linhas de texto no arquivo de configuração. Por exemplo, o pacote de vetor permite que um conjunto de instâncias de protocolo localizadas em cada um dos nós seja tratado como um vetor. Operações com vetores, tais como calcular o ângulo entre dois vetores ou inicializar vetores, são suportados;
- Camada de transporte e alta rotatividade: Para aumentar o realismo da simulação, pode-se configurar para utilizar conjuntos de dados com base em rastreamento. Este recurso é suportado apenas no motor baseado em eventos. A camada de transporte é modelada por meio de um protocolo especial, que proporciona um serviço de envio de mensagem. Modelos de alta rotatividade estão disponíveis para os modelos que têm base em ciclo e evento.

- **SimGrid**: Conforme Casanova, Legrand e Quinson (2008), o projeto foi iniciado em 1999 para permitir o estudo de algoritmos de escalonamento para plataformas heterogêneas. De acordo com a Figura 5, o SimGrid oferece quatro interface de usuário: SimDag é descendente da versão um e foi desenhada para a investigação de heurísticas de escalonamento para aplicações como grafos de tarefas; MSG foi desenhado para trabalhar com processos concorrentes sequenciais. GRAS permite criar aplicações reais distribuídas; e SMPI permite a simulação direta de aplicações MPI.

Figura 5 – Visão geral dos componentes do SimGrid



Fonte: Casanova, Legrand e Quinson (2008)

O módulo XBT é uma *toolbox* utilizada por todo o *software*. SURF é o motor de simulação. SimIX é um módulo interno que provê de serviços POSIX. E SMURF permite a distribuição de processos de simulação sobre um *cluster*, aproveitando a memória de todos os computadores. Os resultados obtidos por Bobelin et al. (2012), em um estudo de simuladores para diversas áreas envolvendo computação paralela e distribuída, demostram que o simulador SimGrid em ordem de grandeza é o mais escalável no estado da arte dos simuladores P2P. Como resultado, foi verificado que ele é quinze vezes mais rápido que o mais rápido utilizado na pesquisa, além de simular cenários que são dez vezes maiores. Essa tendência se mantém quando se utiliza o modelo baseado em fluxo preciso,visto que, durante a simulação, a exatidão é melhorada.

3 TRABALHOS RELACIONADOS

A escolha dos trabalhos relacionados passa para análise de algoritmos e sistemas nas seguintes áreas: (i) Arquitetura - ambiente que caracterizam Grades de *Desktop* e Grades P2P; (ii) Modelo de programação paralela - demonstrar a diversidade utilizada; (iii) Serviços oferecidos - apresentar as preocupações em relação a migração e segurança à falhas; e (iv) Implementação - forma como foi implementada. Além desses requisitos, as soluções deveriam trabalhar em ambientes de alta escala, dinâmicos e heterogêneos. Os trabalhos aqui apresentados refletem uma busca em sites de pesquisa como IEEE Xplore[1], ACM Portal [2], Springer Link [3] e Google Scholar [4] e, no melhor do conhecimento do autor desta pesquisa, representam o estado da arte no que tange ao assunto.

O restante do capítulo está organzado da seguinte maneira: Seção 3.1 - focaliza o modelo DisCoP2P; Seção 3.2 - expõe o modelo Mizan; Seção 3.3 - mostra a solução P3; Seção 3.4 - apresenta um *middleware* baseado em *checkpoint*; Seção 3.5 - apresenta um *middleware* baseado em agentes; Seção 3.6 - mostra um estudo sobre a escalabilidade das técnicas de programação de ciclo dinâmico via simulação discreta; Seção 3.7 e Seção 3.8 - abordam soluções para balanceamento de carga em redes com DHT; Seção 3.9 - trata do escalonamento por roubo de tarefas; Seção 3.10- desenvolve uma solução com *Publish/Subscribe*; e Seção 6.3 - classifica e analisa as soluções.

3.1 Sentís et al. (2014)

DisCoP2P (SENTíS et al., 2014) é a junção das plataformas CodiP2P (CASTELLà et al., 2009) e DisCoP (CASTELLà et al., 2010), que tem como objetivo compartilhar recursos (CPU, memória, etc.) para execução de aplicações paralelas. Esta nova arquitetura toma a vantagem da alta escalabilidade e a eficiência do mecanismo de procura da CodiP2P e a habilidade de classificação de recursos computacionais da DisCoP.

Nesta plataforma, quando um nó necessita executar uma tarefa, ele vira um Mestre. Então, é iniciada a verificação de qual área fará a execução, e, após a identificação, é criado um canal entre o Mestre e o Gestor da área. O Mestre envia a tarefa para o Gestor da área e, após recebida, ela é dividida entre os nós do seu grupo.

As tarefas são criadas nos trabalhadores em um modelo *Round Robin*. Se mais trabalhadores forem necessários, o Gestor de área poderá delegar a divisão das tarefas entre as áreas vizinhas. No entanto, mais de uma tarefa pode ser criada em um trabalhador. As políticas de escalonamento tentam minimizar o número de áreas ocupadas por um trabalho específico. Então, a fragmentação do sistema é minimizada.

[1] http://ieeexplore.ieee.org/Xplore/home.jsp.
[2] http://dl.acm.org/.
[3] http://link.springer.com/.
[4] http://scholar.google.com.br/.

3.2 Khayyat et al. (2013)

Khayyat et al. (2013) examinaram as características de tempo de excução do sistema Pregel (MALEWICZ et al., 2010). O Pregel foi introduzido como um sistema minerador escalável de grafos que pode prover de desempenho significante sobre implementações de MapReduce. A partir desses estudos surgiu Mizan. Mizan é um sistema BSP baseado em grafos, que foca em balanceamento dinâmico e efetivo de carga, validando comunicação e computação por todos os nós executores pertencentes à rede. Primeiramente, são lidos e particionados os grafos de dados por meio dos executores (nós). Então, o sistema procede como uma série de *supersteps*, cada uma separadamente por uma barreira global de sincronização. Durante a *superstep*, cada vértice processa mensagens recebidas da *superstep* anterior e as envia para os vizinhos dos vértices. Mizan balanceia a carga, movendo os vértices selecionados por meio dos trabalhadores. A migração é executada quando todos os executores pesquisam uma barreira de sincronização da *superstep*, para evitar a violação da integridade da computação, isolada e corretamente por meio do modelo BSP.

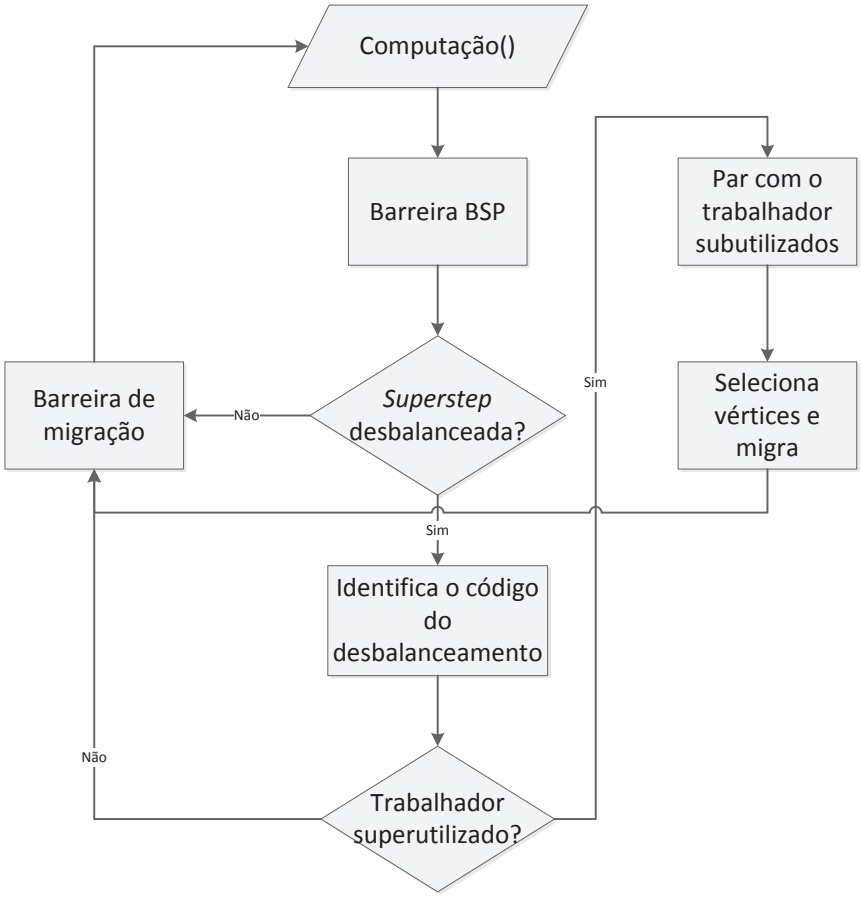
Todos os executores criam e executam o planejamento da migração sem requerer qualquer coordenação central. O planejador de migração é iniciado em todos os executores no fim de cada *superstep* (quando os nós procuram pela barreira de sincronização), depois, é recebido um resumo das estatísticas (números que saíram para outros executores remotos, total de mensagens recebidas e tempo de resposta - tempo execução - durante a *superstep* corrente) de todos os outros executores. De acordo com a Figura 6, é apresentado um resumo do planejamento feito pelo planejador de migração do Mizan, que segue os seguintes passos: (i) identifica o código de falta de balanceamento; (ii) seleciona o objetivo de migração; (iii) cria de pares de sobrecarregados com executores com capacidade; (iv) seleciona o vértice para migração; e (v) faz os vértices migrarem.

Na Figura 7, são demonstrados os módulos que se comunicam entre os executores para atualizar a DHT após a execução da migração dos vértices. Desde que haja migração de vértice na barreira entre duas *supersteps*, todos os executores que têm armazenado a localização do vértice migrado receberão a atualização física do executor inicial da nova *superstep*.

3.3 Shudo, Tanaka e Sekiguchi (2005)

O *middleware* P3 (Shudo, Tanaka e Sekiguchi (2005)) - *Personal Power Plant* - possibilita a transferência mútua e equivalente de poder computacional entre indivíduos, que é proposto pela ideia original do P2P. Qualquer usuário do P3 pode utilizar o recurso de outros para submeter uma tarefa que necessite de computação. Também é suportada distribuição de alta escala utilizando computadores heterogêneos. Pelo *middleware* JXTA, um usuário de recurso cria um grupo de tarefas para sua tarefa e publica um anúncio para o grupo, conforme Figura 8. Provedores de recursos descobrem o grupo da tarefa, verificam se ele deseja colaborar para a mesma

Figura 6 – Resumo do planejamento de migração - Mizan



Fonte: Khayyat et al. (2013)

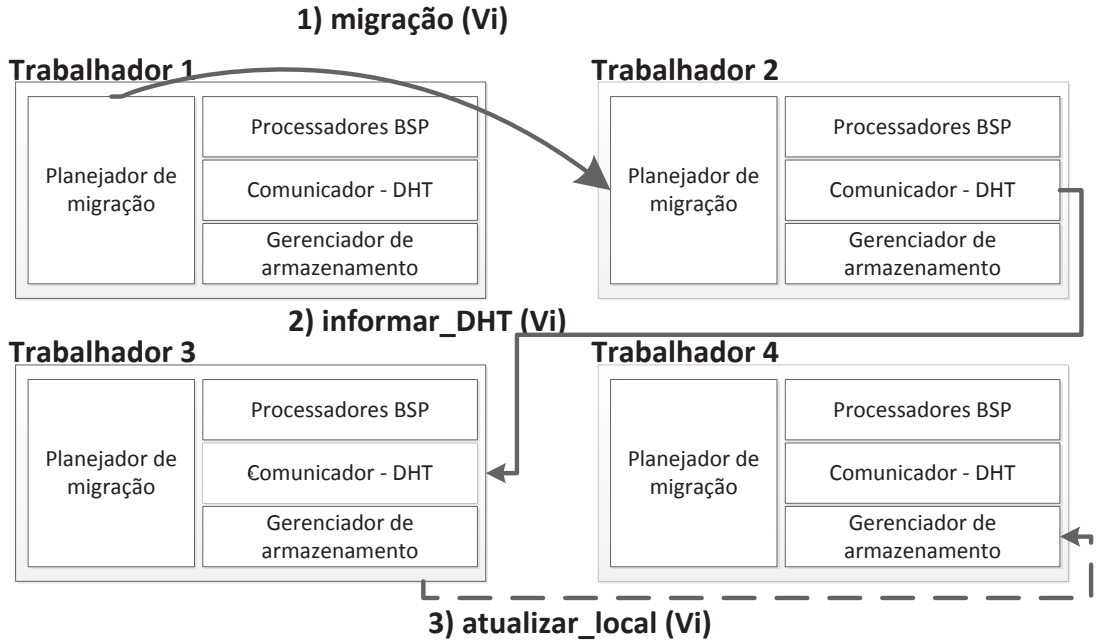
e se juntam ao grupo que a executará. Todo computador também é descoberto do mesmo modo que uma tarefa.

Por meio da biblioteca *master/worker* (mestre/trabalhador), tarefas de um determinado trabalho computacional são entregues e escalonadas. Um trabalhador pode inicializar ou finalizar uma tarefa a qualquer momento, mesmo que ela já tenha iniciado. No *middleware*, esses eventos foram controlados, não havendo necessidade de controle pelo desenvolvedor.

3.4 Camargo, Castor e Kon (2010)

Camargo, Castor e Kon (2010) apresentam um *middleware* que permite distribuição confiável de armazenamento de dados da aplicação nas máquinas compartilhadas em um ambiente redundante e com tolerância à falhas. Um mecanismo baseado no *checkpoint* monitora a execução das aplicações paralelas, que salva periodicamente *checkpoints* nas máquinas compartilhadas, e, em caso de falhas nos nós, é suportada a migração da aplicação por meio dos nós heterogêneos na Grade. Para permitir o armazenamento e o gerenciamento das entradas e saí-

Figura 7 – Vertex de migração

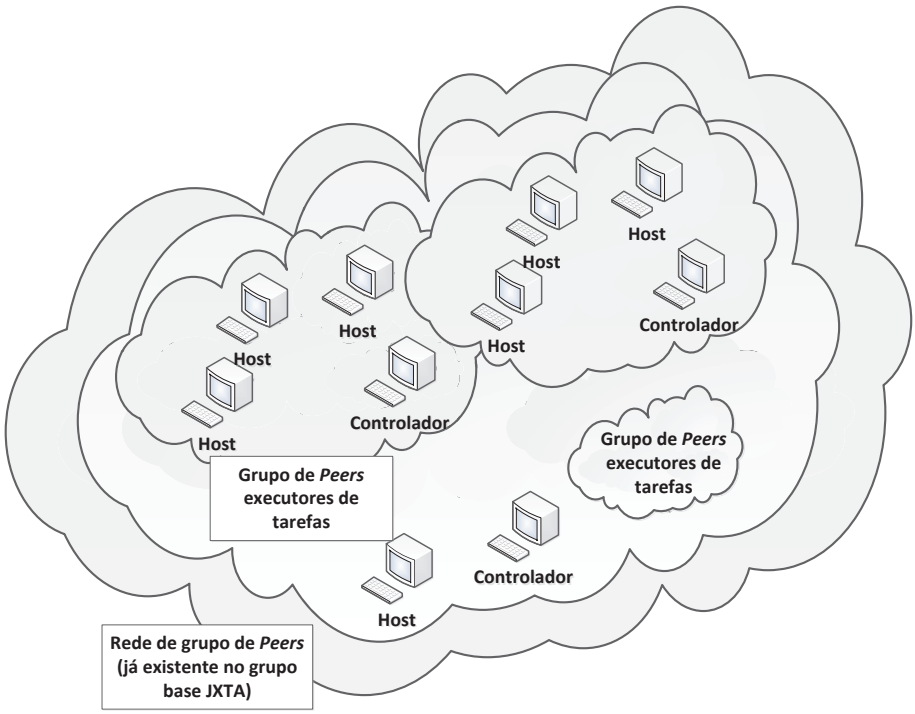


Fonte: Khayyat et al. (2013)

das da aplicação, dados de *checkpoint*, a arquitetura foi organizada em *Cluster*, conectada por uma rede P2P tolerante à falhas e auto-organizável. No *middleware*, são armazenados dados de uma maneira distribuída e são codificados em fragmentos redundantes, permitindo a reconstrução original utilizando somente as partes que foram armazenadas. Os dados armazenados podem ser acessados de qualquer máquina na Grade e fragmentados para serem baixados de outras máquinas com conexões rápidas. Para administrar essa heterogeneidade de recursos, foi utilizado o protocolo de roteamento P2P *Pastry* para suportar IDs virtuais (chave *hash* que o nó recebe), que permitem a seleção dos *Clusters* contendo máquinas com alta disponibilidade de armazenamento de dados.

Para oferecer tolerância à falhas, o algoritmo possui um mecanismo de *checkpoints* para *rollback* de recuperação, que salva a condição em *checkpoint* e, caso haja falhas no nó, reinicializa a aplicação do estado contido no último *checkpoint* salvo. O mecanismo trabalha para sequenciais, *bag-of-tasks*, e BSP em aplicações acopladas paralelamente. Mas, no caso de utilização BSP, é necessário garantir que a geração dos *checkpoints* seja consistente. Por isso, foi utilizado um modelo BSP chamado BSPlib (HILL et al., 1998), que contém funções extras que podem ser utilizadas para controle da geração de *checkpoint*. Sendo assim, a criação deste *middleware* permite que instituições utilizem recursos ociosos de máquinas já existentes para executar computação de alto desempenho e armazenar dados de maneira confiável. A execução confiável paralela de equipamentos não dedicados é disponiblizada por meio de um mecanismo de *checkpoint* com baixo custo.

Figura 8 – Organização do *software* de gerenciamento e relacionamento entre grupos de *peers*



Fonte: Shudo, Tanaka e Sekiguchi (2005)

3.5 Son, Lee e Youn (2010)

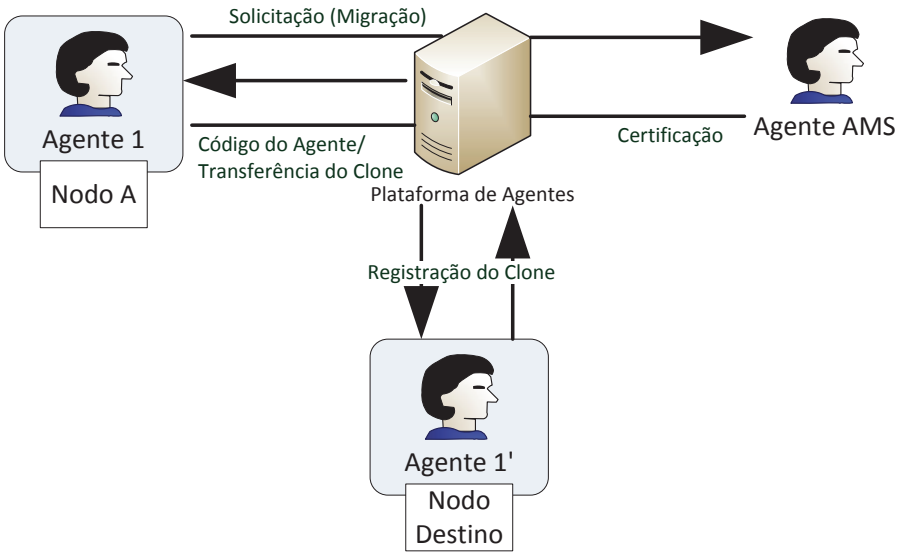
No trabalho apresentado por Son, Lee e Youn (2010), é exposto um sistema multiagente para ambientes de computação ubíquos que pode prover de serviços customizados os usuários para, efetivamente, utilizar os recursos distribuidamente. As abordagens existentes de balanceamento dinâmico de carga invocam a migração de agentes, mesmo em períodos de desequilíbrio de carga. Para solucionar isso, é proposto um esquema baseado em predição para balanceamento dinâmico de carga. O balanceamento de carga é requerido, caso um agente específico esteja sobrecarregado ou a comunicação entre os agentes existentes seja excessiva. O serviço com base em agentes é suportado pela colaboração de vários agentes. Dessa maneira, a carga de trabalho de um agente necessita ser determinada com o número de outros que colaborarão. A carga de trabalho de cada um é armazenada e controlada pelo agente de balanceamento. O controle é previsto por meio da média exponencial, usando os dados armazenados.

A técnica utilizada para o desenvolvimento deste algoritmo é baseada em agente, em que o mesmo é um programa voluntário que, automaticamente, gerencia as tarefas em favor do cumprimento de um objetivo imposto por algum usuário. Ele possui autonomia para tomar decisões próprias, sem alguma ordem ou interferência, e, além disso, tem inteligência para planejar por meio das intenções do usuário, utilizando uma base de conhecimento, e também possui habilidade de entendimento enquanto atualizada a base de conhecimento pelo aprendizado. Com a colaboração entre eles, atendem a um objetivo comum e, com a habilidade de se socializarem,

conseguem assistência de outros agentes para tarefas difíceis, manuseando-as independentemente.

O sistema multiagente proposto emprega sistema de agentes móveis para suportar a sua migração. Para a migração, ele lida com mudanças dinâmicas do estado da carga, não somente com o estado do sistema, mas, também, com os agentes operantes e registrados. O nó destino tem que ter mais de um agente executando. O agente solicita a migração com o envio de uma mensagem para o agente de migração (AMS). Então, o AMS envia uma mensagem de autenticação. O agente envia uma mensagem de solicitação para o DLA (*Dynamic Library Agent*) do nó destino. O DLA, então, envia o código do agente após verificar a autenticação e a validação da mensagem. Finalmente, o agente faz migrar a si próprio para o nó destino ou migra um clone seu. O agente migrado é executado pela DLA, e, se a migração é um clone, ele é registrado na plataforma, conforme pode ser visto na Figura 9.

Figura 9 – Procedimeto do agente de migração



Fonte: Son, Lee e Youn (2010)

Essa estrutura foi dimensionada para selecionar um nó sobrecarregado e migrar o agente depois de coletar as informações de carga de cada nó, alocando-o no agente de balanceamento de carga. Isso é uma abordagem centralizada. Com a coleção de carga centralizada, a do agente de balanceamento rapidamente aumenta conforme o crescimento do número de agentes, então a velocidade de processamento diminui. O nó muda de estado periodicamente.

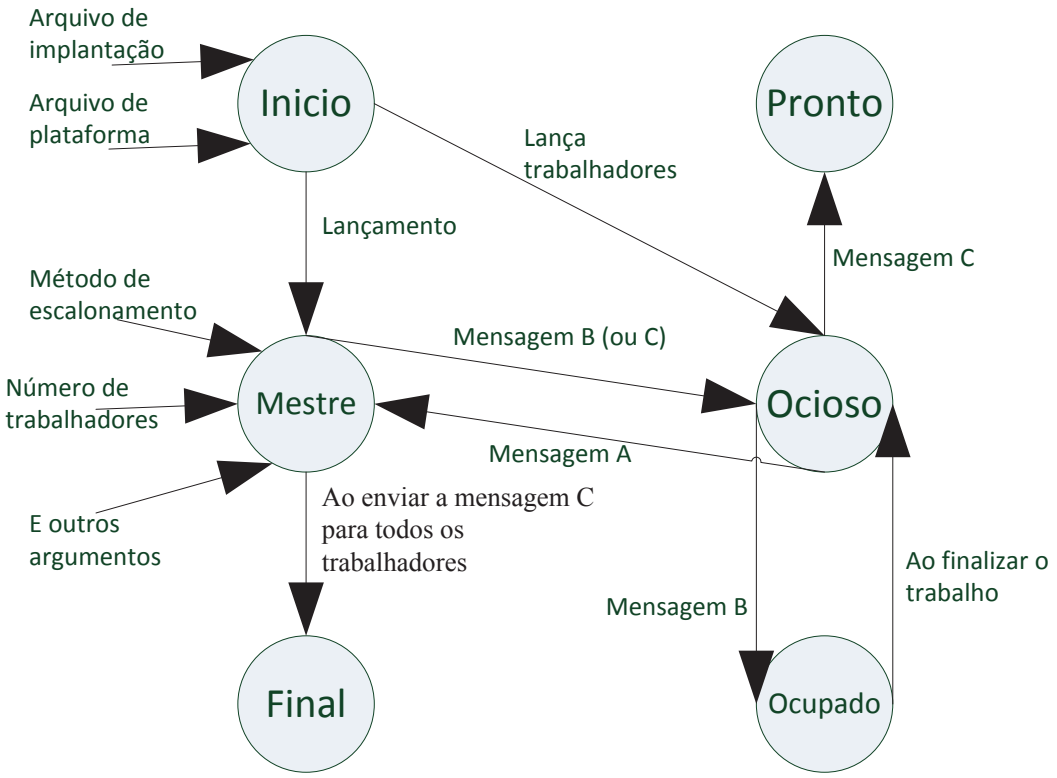
3.6 Balasubramaniam et al. (2012)

Conforme Balasubramaniam et al. (2012), propõe-se um estudo sobre a escalabilidade das técnicas de programação de ciclo dinâmico via simulação discreta, tanto em termos de número

de processadores, quanto do tamanho do problema. Para facilitar o estudo de escalabilidade, um escalonador de ciclo dinâmico foi projetado e implementado, usando o ambiente de simulação SimGrid. Com os anos, um número de técnicas de programação de ciclo dinâmico tem sido desenvolvido. Essas técnicas são fundamentadas em análises probabilísticas e são efetivas para enfrentar os desequilíbrios de carga imprevisíveis no sistema decorrentes de várias fontes, tais como variações na aplicação, algoritmos e características sistêmicas.

Conforme a Figura 10, é demonstrado a estruturação da simulação no SimGrid. Ela foi implantada para execução de um aplicação mestre/escravo, onde cada *host* executa uma função específica, podendo ser mestre ou escravo. Para a função de mestre outros argumentos são exigidos (por exemplo, o método de programação, número de trabalhadores, número de tarefas, arquivo de entrada que descreve o tempo de execução das tarefas, etc.) estão disponíveis como argumentos de linha de comando. A partir do arquivo de entrada e após a leitura dos tempos de execução das tarefas, o mestre estabelece a fila de tarefas que estão prontas para serem processadas. O escalonador criado é responsável pela atribuição de tarefas ditadas pelas regras do algoritmo de escalonamento.

Figura 10 – Arquitetura do escalonador de ciclos dinâmicos



Fonte: Balasubramaniam et al. (2012)

3.7 Godfrey et al. (2004)

Este tipo de topologia é amplamente disseminado na criação de redes P2P e, devido a isso, muitas pesquisas relacionadas ao balanceamento deste tipo de estrutura são frequentes, como, por exemplo, Godfrey et al. (2004), que apresentam um algoritmo para balanceamento de carga em sistema P2P heterogênea e dinâmica. A maioria dos sistemas P2P oferece uma DHT, a qual permite que haja abstração da distribuição randomica dos objetos distribuídos nos "nós/*peer*". O desbalanceamento pode ter resultado da não uniformidade da distribuição dos objetos no espaço de identificação e de um alto grau de heterogeneidade nas cargas dos objetos e nas capacidades dos nós. A carga de um nó pode ter grande variação através do tempo devido a contínuas inserções e a chegadas e partidas dos nós.

No algoritmo proposto, são utilizados conceitos de servidores virtuais que representam os nós na DHT. Uma das dificuldades de balanceamento de carga em DHT é que o balanceador tem pequeno controle sobre onde os objetos estão armazenados. A maioria das DHT utilizam *consistent hashing* para mapear os objetos nos nós: ambos os objetos e nós no sistema possuem uma identidade única denominada ID no mesmo espaço de identificação, e um objeto está armazenado em um nó com o mais próximo ID no espaço. Essa associação com cada nó em uma região de identificação demonstra pelo que ele é responsável. Uma das maiores vantagens de se utilizarem servidores virtuais para balanceamento de carga é que, a partir dessa abordagem, não são necessárias mudanças na DHT básica. Assim, a transferência dos servidores virtuais pode ser simplesmente implementada como a partida de um *peer* ou entrada de um *peer* no sistema.

O balanceador de carga proposto para uma rede estruturada com Chord utiliza a ideia básica de armazenamento de informação de um nó *peer* em certo número de diretórios que, periodicamente, escalonam a redesignação das máquinas virtuais para alcançar o melhor balanceamento de carga, essencialmente reduzindo a dificuldade de balanceamento de carga distribuída para um problema central em cada diretório. Cada diretório possui um ID conhecido por todos os nós e é armazenado no nó responsável por aquele ID. Cada nó, inicialmente, escolhe um diretório e utiliza o protocolo de procura da DHT para reportar ao diretório a carga dos servidores virtuais. Cada diretório coleta informações de carga e da capacidade dos nós contatados. Periodicamente, é computada a carga para escalonamento das transferências dos servidores virtuais em torno dos nós com o objetivo de reduzir a sua utilização máxima.

3.8 Wu, Tian e Ng (2006)

Conforme Wu, Tian e Ng (2006), há um problema fundamental em sistemas P2P baseados em DHT: balanceamento de carga é importante para evitar a degradação de desempenho e garantir equidade do sistema. Devido à limitação do balanceamento de carga em *namespace* pura, servidores virtuais permitem a migração entre os nós. Com a mudança de um número apropriado de servidores virtuais de nós pesados para nós leves, se obtém o balanceamento de carga.

Com fundamento nas diferenças nas informações de gerenciamento de carga e decisões feitas pelo balanceamento, as abordagens baseadas em migração podem ser caracterizadas em duas estratégias: Estratégia de Diretório de Encontro (*Rendezvous Directory Strategy* - RDS)(BYERS; CONSIDINE; MITZENMACHER, 2003) e Estratégia de Procura Independente (*Independent Searching Strategy* - ISS)(NAOR; WIEDER, 2003). Na estratégia RDS, o diretório que é atualizado periodicamente é responsável pelo escalonamento da redesignação de carga para alcançar o balanceamento, porém a entrada e saída de nós na rede (devido aos nós atualizarem os diretórios na sua primeira entrada e enviarem atualizações periódicas) podem causar problemas de informações supérfluas e prejudicar o desempenho da estratégia. Na estratégia ISS, os nós não publicam suas informações de carga, somente as disponibilizam quando requisitadas. Para alcançar o balanceamento de carga, um nó deve executar uma consulta (por meio de amostra) independente para encontrar outros nós com capacidade e, então, fazer migrar a carga de um nó pesado para um leve.

Por meio das características do RDS, é proposta a solução Estratégia Baseada em Fofoca (*Gossip-Based Strategy* - GBS), que foi desenhada para explorar a eficiência da RDS enquanto fornece escalabilidade e robustez. A GBS foi criada no topo de um anel DHT, assim como o Chord, em que, baseados no tamanho do sistema, um ou mais grupos de balanceamento de carga são formados em torno dos *peers*. Para sistemas com poucos milhares de *peers*, somente é formado um grupo, mas, para sistemas com centenas de milhares, os *peers* formam múltiplos grupos, cada um correspondente a uma região contínua com tamanho igual no anel. Todos os *peers* que estão em um mesmo grupo compartilham o mesmo prefixo, que se refere como o ID do grupo. Na GBS, em vez de cada nó se reportar a um diretório fixo, eles disseminam informações de carga dentro de seu grupo por um protocolo de fofoca de forma a reduzir o tráfego de mensagens, pois uma "árvore" de fofoca firmada na DHT é utilizada para disseminar as informações. Essa árvore não requer manutenções explícitas e apenas se expande com base em informações locais. Cada nó publica informações como: sua capacidade, a carga de todas as máquinas virtuais e o seu endereço IP.

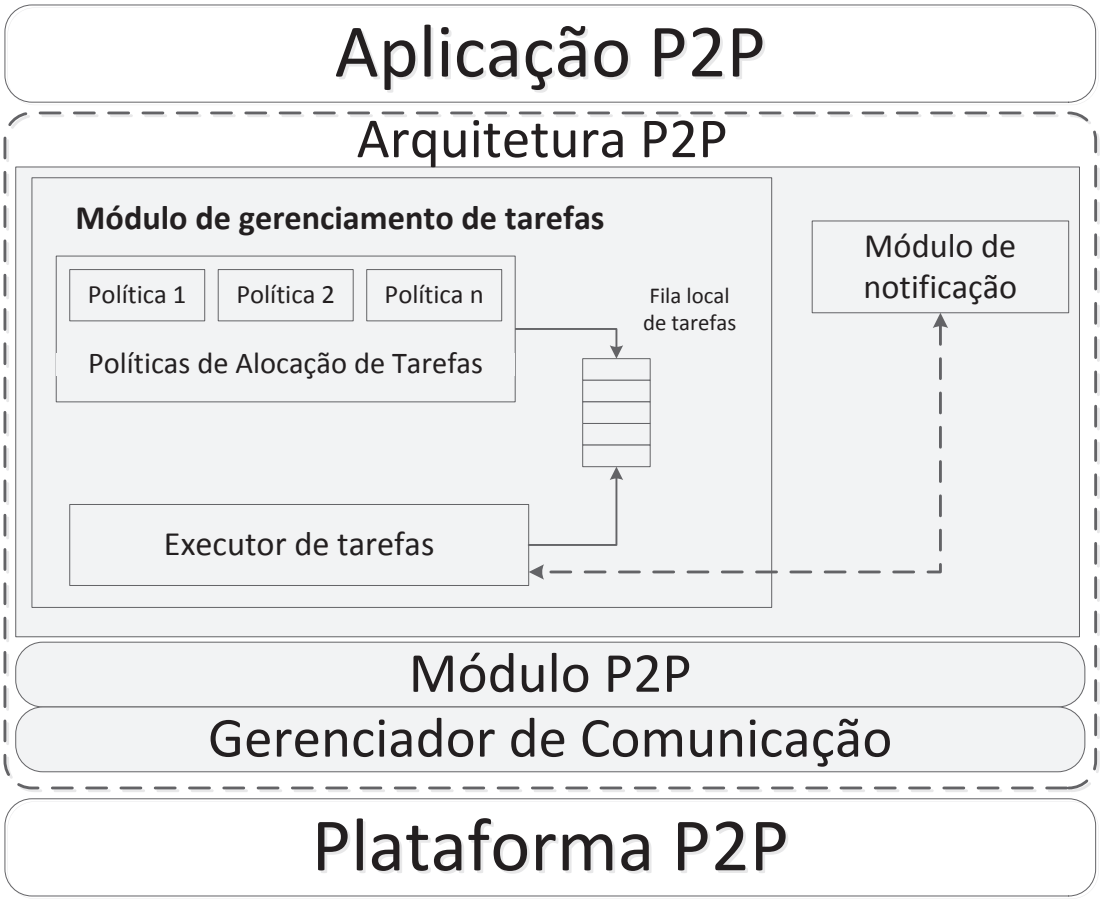
3.9 Leite et al. (2012)

Leite et al. (2012) desenvolveram uma arquitetura flexível para Grades de *desktop* que suporta políticas de múltiplas alocações de tarefas. Uma aplicação *Bag-of-Tasks* é submetida para nós randômicos e alocada em suas filas locais, que são executadas no modo FIFO. Quando um equipamento estiver ocioso, uma política de alocação de tarefas será executada e trará tarefas de nós remotos em uma arquitetura que foi implementada no topo de um *middleware* JXTA, usando um algoritmo de busca na *overlay* Chord. Quando uma fila local está vazia e a política de roubo de trabalho está ativa, uma tarefa é roubada de forma randômica de um nó com uma fila não vazia. Diferentemente das políticas tradicionais de roubo de carga, o nó original também faz manutenção das tarefas na sua fila. Portanto, mais do que uma cópia da tarefa pode

existir no sistema. A arquitetura mantém o curso das execuções de cada tarefa de uma maneira descentralizada e ela é considerada acabada quando sua primeira réplica completa a execução.

A arquitetura é composta por quatro módulos (Figura 11): Gerenciador de comunicação, Módulo P2P, Módulo de Gerenciamento de Tarefas e Módulo de Notificação. Essas são independentes da topologia de rede da plataforma P2P. Dentre os módulos citados, o responsável pelo controle e execução das réplicas e das tarefas é o Módulo de Execução de Tarefas. Ele trabalha de maneira FIFO, uma tarefa por vez é executada, antes que a fila esteja vazia. Quando a fila local estiver vazia, a política de roubo de carga poderá ser acionada. Desde que muitas réplicas da tarefa *t* possam existir no sistema, o nó que contenha uma réplica da tarefa *t* é notificado pelo nó que submeteu quando finalizada. Quando um nó recebe tal notificação, ele apaga a tarefa de sua fila local.

Figura 11 – Arquitetura P2P de alocação de tarefas



Fonte: Leite et al. (2012)

3.10 Anceaume et al. (2006)

Anceaume et al. (2006) utilizam o sistema *Publish/Subscribe*. Devido a este oferecer uma

plataforma para a entrega de dados (eventos) de publicadores para assinantes em uma forma anônima em redes distribuídas. Nesta pesquisa, é apresentado um sistema *Publish/Subscribe* baseado em conteúdo chamado DPS (*Dynamic Publish/Subscribe*) com características de *Self-** que foi inserido em uma rede P2P estruturada. Foram apresentados diferentes métodos de assinatura e para publicadores que podem ser combinados para obter quatro diferentes implementações do sistema: *Leader*, *Epidemic*, *Root-based* e *Generic*. Pelas simulações, foi concluído que a abordagem *Leader* é mais agradável para um conjunto pequeno de equipamentos que estão menos inclinados a erros. A abordagem *Epidemic* disponibiliza maior segurança, melhor escalabilidade e balanceamento de carga no custo de mensagens mais complexas. *Generic* é mais confiável para processos de assinaturas, pois melhor distribui a carga. E *Root-based* oferece menor latência. Como o sistema proporciona a seleção de diferentes aplicações, torna-se versátil.

3.11 Rosa Righi et al. (2011)

MigBSP é um modelo de reescalonamento que funciona através de recursos heterogêneos, unindo o poder de *clusters*, supercomputadores e redes locais. A questão da heterogeneidade considera relógio dos processadores (todos os processadores têm o mesmo conjunto de instruções), bem como a largura de banda da rede. Tal arquitetura é montada com Conjuntos (*sites*) e Gestores de Grupos. Os Gestores de Grupos são responsáveis pelo escalonamento, capturando os dados de um Conjunto e trocando-os entre os outros gestores.

A decisão para o processo de remapeamento é tomada no final de uma *superstep*. Com o objetivo de gerar o mínimo possível de intromissão na aplicação, são duas adaptações que controlam o valor de $\alpha(\alpha \in \mathbb{N}^*)$. α é atualizado a cada chamada de reescalonamento e indicará o intervalo para a seguinte. Os objetivos das adaptações são: (i) adiar a chamada reprogramação caso os processos estejam em equilíbrio ou para transformá-los mais frequentes, caso contrário; (ii) adiar essa chamada, caso haja um padrão sem migrações nas últimas ω chamadas. Uma variável chamada D é usada para indicar uma porcentagem de quão longe os processos mais lentos e os mais rápidos podem ser a partir da média para considerá-los como balanceados.

Há a resposta para o "que"é resolvido através da função de decisão chamada Potencial de Migração (PM). Cada processo i calcula n funções $PM(i,j)$, onde n é o número de conjuntos e j um conjunto. A chave do raciocínio consiste na realização de apenas um subconjunto no momento do reescalonamento dos testes de processos de recursos. $PM(i,j)$ utiliza como métricas Computação, Comunicação e Memória, como pode ser visto nas Equações 3.1, 3.2, 3.3 e 3.4. Quanto maior for o valor de $PM(i,j)$, o mais propenso a processos será migrar.

$$Comp(i,j) = P_{Comp}(i).CTP(i).ISet(j) \tag{3.1}$$

$$Comm(i,j) = P_{Comm}(i,j).BTP(i,j) \tag{3.2}$$

$$Mem(i,j) = M(i).T(i,j) + Mig(i,j) \tag{3.3}$$

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \tag{3.4}$$

A métrica computação - $Comp(i,j)$ - considera um padrão de Computação $P_{Comp}(i)$, que mede a estabilidade de um processo i em relação à quantidade de instruções em cada *superstep*. Esse valor é próximo de 1, caso o processo seja regular, e próximo de 0, caso contrário. Essa métrica também realiza uma previsão tempo de computação ($CTP(i)$) para cada processo i com base em todas as fases de computação entre as duas ativações de reescalonamento. $Comp(i,j)$ também apresenta um índice $ISet(j)$, que informa a capacidade média de Conjunto j. Da mesma forma, a métrica Comunicação - $Comm(i,j)$ - calcula o padrão de comunicação $P_{Comm}(i,j)$ entre processos e conjuntos. Além disso, essa métrica usa a predição de tempo em comunicação $BTP(i,j)$, considerando dados entre duas ativações de reequilíbrio. $Comm(i,j)$ aumentará se caso o processo i tiver uma comunicação regular com os processos de conjunto j e realizar ações de comunicação mais lentas para esse conjunto. Métrica Memória - $Mem(i,j)$ - considera a memória do processo, a taxa de transferência entre o processo considerado e o gerente do conjunto-alvo, bem como os custos de migração. Esses custos são dependentes do sistema operacional, assim como da ferramenta de migração.

A cada chamada de reescalonamento, o processo passa seu mais alto $PM(i,j)$ para o seu Gestor de Conjunto. A última entidade troca o PM dos processos entre os outros Gestores. Cada gestor cria uma lista ordenada decrescente selecionando o superior para testar a viabilidade de migração. O teste considera os seguintes dados: (i) a carga externa na fonte e processadores de destino; (ii) os processos que ambos os processadores estão executando; (iii) a simulação de processo em execução considerando um processador de destino; (iv) o tempo de ações de comunicação, considerando processadores locais e de destino; (v) custos de migração.

3.12 Análise

Pela busca do aproveitamento de recursos computacionais, soluções criativas surgem. Os seus objetivos geralmente fundamentam-se na criação de ambientes de forma que os recursos atuem juntos. Através da colaboração dos recursos é criado uma ambiente que auxilia na solução de problemas complexos, isso porque aumenta capacidade computacional. Na Tabela 3, é demonstrado que não há um modelo fixo para a criação de ambientes colaborativos. Além disso, não existe fórmula fixa para escalonamento, e as soluções têm diversidades em relação à arquitetura, ao modelo de programação paralela, aos serviços oferecidos e aos métodos de avaliação.

Tabela 3 – Avaliação entre estruturas e algoritmos de balanceamento

Trabalho	Arquitetura	Modelo de Programação Paralela	Serviços	Método	Observação
Sentís et al. (2014)	Grades P2P	Mestre/Escravo	Migração	Métricas Computacionais	Computação e Memória
Khayyat et al. (2013)	Grades de *Desktop*	BSP	Migração	Métricas Computacionais	Computação
Shudo, Tanaka e Sekiguchi (2005)	Grades P2P	Mestre/Escravo	Migração	Métricas Computacionais	Computação
Camargo, Castor e Kon (2010)	Grades P2P	BSP	Migração e Replicação de dados	Métricas Computacionais	Computação
Son, Lee e Youn (2010)	Grades de *Desktop*	Mestre/Escravo	Migração	Métricas Computacionais	Computação
Balasubramaniam et al. (2012)	Grades de *Desktop*	*Bag of Tasks*	Não Informado	Métricas Computacionais	Computação
Godfrey et al. (2004)	Grades P2P	Mestre/Escravo	Migração e Replicação de Dados	Métricas Computacionais	Computação
Wu, Tian e Ng (2006)	Grades P2P	Mestre/Escravo	Migração e Replicação de Dados	Métricas Computacionais	Computação
Leite et al. (2012)	Grades P2P	*Bag of Tasks*	Migração e Replicação de Computação	Política de Roubos de Trabalhos	Computação
Anceaume et al. (2006)	Grades P2P	*Bag of Tasks*	Não Informado	Métricas Computacionais	Computação
Rosa Righi et al. (2011)	Grade de Computadores	BSP	Migração	Métricas Computacionais	Computação, Comunicação e Memória

Fonte: Elaborado pelo autor

Conforme pode ser visto na tabela, o foco principal dos trabalhos tem como objetivo utilizar tarefas independentes (não possuam relação). Isso fica claro a partir dos modelos de programação paralelos utilizados (Mestre/Escravo e *Bag-of-Tasks*). Entre os modelos apresentados, somente Rosa Righi et al. (2011), Camargo, Castor e Kon (2010) e Sentís et al. (2014) utilizaram um modelo de programação voltado para tarefas dependentes, no caso, o BSP.

Além disso, ao avaliar os modelos que utilizam arquiteturas do tipo BSP com Grades P2P, fica evidente que há escassez entre a combinação desses métodos. Somente os modelos propostos por Khayyat et al. (2013) e Camargo, Castor e Kon (2010) as utilizaram. Isso acontece devido a existência de grande dificuldade de controle sobre a rotatividade dos participantes, além do aumento da complexidade de controle sobre o escalonamento para balanceamento de carga.

Outro ponto é que poucos modelos possuem escalonamento com visão global do ambiente. Ao avaliar as métricas computacionais utilizadas para o balanceamento de carga, é verificada pouca preocupação com métricas não envolvendo computação. Nesse requisito dois modelos estão em destaque: (i) o modelo Sentís et al. (2014), que utilizou a métrica Memória em suas avaliações; e (ii) Rosa Righi et al. (2011) preocupou-se de forma mais global inserindo métricas como Comunicação e Memória.

Outro ponto evidente é a preocupação em reavaliar o ambiente afim de fazer migrar os processos de um recurso lento para outro com maior velocidade. Entre os modelos apresentados, a sua maioria possuía esse tipo de serviço. Porém, poucos tinham algum controle de falhas. Os modelos Leite et al. (2012), Wu, Tian e Ng (2006), Godfrey et al. (2004) e Shudo, Tanaka e Sekiguchi (2005) apresentaram ferramentas de replicação de dados para correção de falhas.

Analisando a Tabela 3, são verificados os seguintes espaços para desenvolvimento de pesquisa:

- Uso do modelo de programação BSP em ambientes e Grades P2P. As soluções geralmente utilizam tarefas independentes, e dessa forma, modelos como Mestre/Escravo e *Bag of Tasks* são comuns. Ao se trabalhar com tarefas dependentes é necessário maior gestão e controle, o que aumenta a complexidade de implementação. Sendo assim, o modelo BSP atende a essa complexidade e é de fácil implementação;

- Uso de migração em ambientes de Grades P2P com o modelo de programação BSP não é comum. As características deste tipo de aplicação geram grande ganho em ambientes dinâmicos, heterogêneos e de alta escala. Isso ocorre porque existe a dinamicidade das entradas e saídas da rede, o que não gera garantias da permanência dos usuários;

- Uso das barreiras do BSP para reescalonamento e *checkpoints*. A partir de avaliações em certas barreiras do BSP será feita a avaliação do ambiente. Isso será realizado para evitar sobrecarga devido à constante comunicação;

- Uso de uma arquitetura híbrida para o ambiente. Ambientes estruturados e não estruturados são pouco utilizados de maneira combinada. Nas soluções, a grande maioria utiliza um dos tipos.

A partir dos espaços verificados, surgiu o modelo BSPonP2P, o qual objetiva aprimorar o uso computacional de equipamentos em ambientes de computação em Grades P2P que utilizam aplicações BSP. A sua arquitetura utiliza dois níveis: (i) Nível de gestores (rede estruturada). Neste nível, somente os gestores são conectados. Este tipo de estratégia foi feito para ter alta escalabilidade e facilitar a comunicação; (ii) Nível de grupo de nós (rede não estruturada). Neste nível, são encontrados os nós, que criam necessidades de execuções e realizam as tarefas. Além disso, como o modelo utiliza uma forma adaptativa de balanceamento, a estratégia é trabalhar em cima do reescalonamento. Nesse ponto, é possível avaliar o ambiente para gerar migrações e balancear a carga.

4 BSPONP2P: GRADES P2P PARA REESCALONAMENTO DE PROCESSOS EM AMBIENTES BSP

Considerando o avanço da *Internet* e a facilidade para adquirir recursos computacionais (no caso, computadores pessoais, *tablets*, *smartphones* e *notebooks*), é possível inferir que haja ociosidade nesses equipmanentos. Isso porque os equipamentos são utilizados, na sua grande maioria, para acesso a redes sociais, consultas à *Internet* e programas que consumem pouco recurso computacional (KIJSIPONGSE; U-RUEKOLAN, 2013). Sendo assim, a fim de aproveitar essa ociosidade, é apresendo o modelo BSPonP2P. O BSPonP2P utiliza aplicações BSP em redes de Grades P2P para computação colaborativa, o que é dado a partir da criação de um ambiente que possui uma arquitetura *overlay*, bem como constantes avaliações do ambiente. Esse ambiente visa a diminuir o desbalanceamento de carga dos processos, reduzindo o tempo de execução de cada *superstep*. Para atender a esse objetivo, é analisado o tempo de execução, migração e comunicação em determinadas *supersteps*.

Neste capítulo, é apresentado o modelo BSPonP2P e as ideias envolvendo arquitetura e reescalonamento. Ele está organizado da seguinte maneira: Seção 4.1 - apresenta as decisões referentes às escolhas em relação a técnicas, modelo, abrangência, escala e estratatégias. Seção 4.2 - demonstra as classificações referentes à estruturação do modelo de Grade de Computação *Desktop* nas quais o BSPonP2P está incluído. Seção 4.3 - trata das definições de terminologias, arquitetura e roteamento. Seção 4.4 - expõe as heurísticas de escalonamento e reescalonamento. Seção 4.5 - aborda os recursos de controle que o modelo possuo, como migração e *checkpoint*.

4.1 Decisões de Projeto

O projeto teve vários pontos decisivos na sua estruturação. A Tabela 4 apresenta as decisões de projeto e as argumentações de cada escolha. Sendo assim, pontos relevantes devem ser atendidos, como: a escala da quantidade de usuários, controles que o modelo precisa executar, recursos que precisa ter para conseguir eficiência e também como será executado.

Decisão	**Solução**	**Motivo**
Arquitetura	Grade P2P	A decisão de utilizar o modelo Grades P2P foi devido a ele combinar P2P com Grades de *Desktop*. Essa combinação facilita a gestão do aproveitamento dos recursos ociosos de computadores pessoais, o que é frequente na realidade em que se vive. Por isso, este tipo de solução tem destaque em comparação à Computação em Nuvem e ao trabalho individualista dos modelos de Grades de Computadores e P2P;
Estrutura	Dois níveis: Estruturada e Desestruturada	Ambientes voluntários de compartilhamento computacional são redes de alta escala porque têm restrições relativamente altas de número de usuários. A utilização de redes estruturadas com DHT possui segurança e agilidade no encontro de recursos e troca de mensagens em redes de alta escala. Redes desestruturadas têm grande flexibilidade por trabalharem em ambientes dinâmicos, além de conseguirem armazenar maior número de informações sobre o ambiente;

Continua na próxima página

Tabela 4 – *Continuação da página anterior*

Decisão	**Solução**	**Motivo**
Troca de Mensagens	MOM	O escalonamento e reescalonamento são baseados na capacidade dos nós, e, sendo assim, o Gestor de Grupo (*Broker*) deve possuir informações atualizadas dos nós. Para atender a esses requisitos, o modelo MOM foi selecionado. MOM gere ambientes dinâmicos a partir da criação de filas na troca de mensagens e trabalha de maneira rápida;
Modelo de programação paralela	BSP	Os processos de um trabalho podem ter ou não dependência entre si. O ambiente mais complexo que pode ser citado entre os dois é o que possua dependência e, dessa forma, o modelo tratará o ambiente com processos dependentes. O modelo de programação BSP atende a essa necessidade por completo por sincronizar os processos nas barreiras de seu modelo;
Ponto de controle para reavaliação do ambiente	*Superstep*	Para que haja o reescalonamento é necessário determinar um ponto no modelo no qual os processos estejam sincronizados e parados. Na *superstep*, os processos que foram executados mais rapidamente estarão esperando os mais lentos. Quando os mais lentos estiverem finalizados, o sistema, antes de iniciar a próxima *superstep* fará a reeavaliação do ambiente;
Tolerância a falhas	Replicação de dados e *Snapshot - Checkpoint*	No ambiente P2P, é necessário algum controle sobre algumas falhas que possam acontecer: no caso, saídas inesperadas de usuários da rede. Para isso, o modelo fará pontos de recuperação para não haver grandes perdas e esse ponto estará situado junto à reavaliação do sistema. Como há grande volume de dados nesses pontos de recuperação, eles serão divididos e replicados entre os participantes da rede;
Recursos adicionais	Migração	O ambiente Grade P2P é heterogêneo e dinâmico, e, sendo assim, a ociosidade pode variar constantemente nesse ambiente. Recursos computacionais que possuam maior capacidade de execução podem surgir e os processos podem estar sendo executados em equipamentos com pouco recurso. O modelo utiliza a técnica de migração para otimizar o tempo de execução, retirando o processo do equipamento com menor poder computacional e migrando para o que possua maior poder.

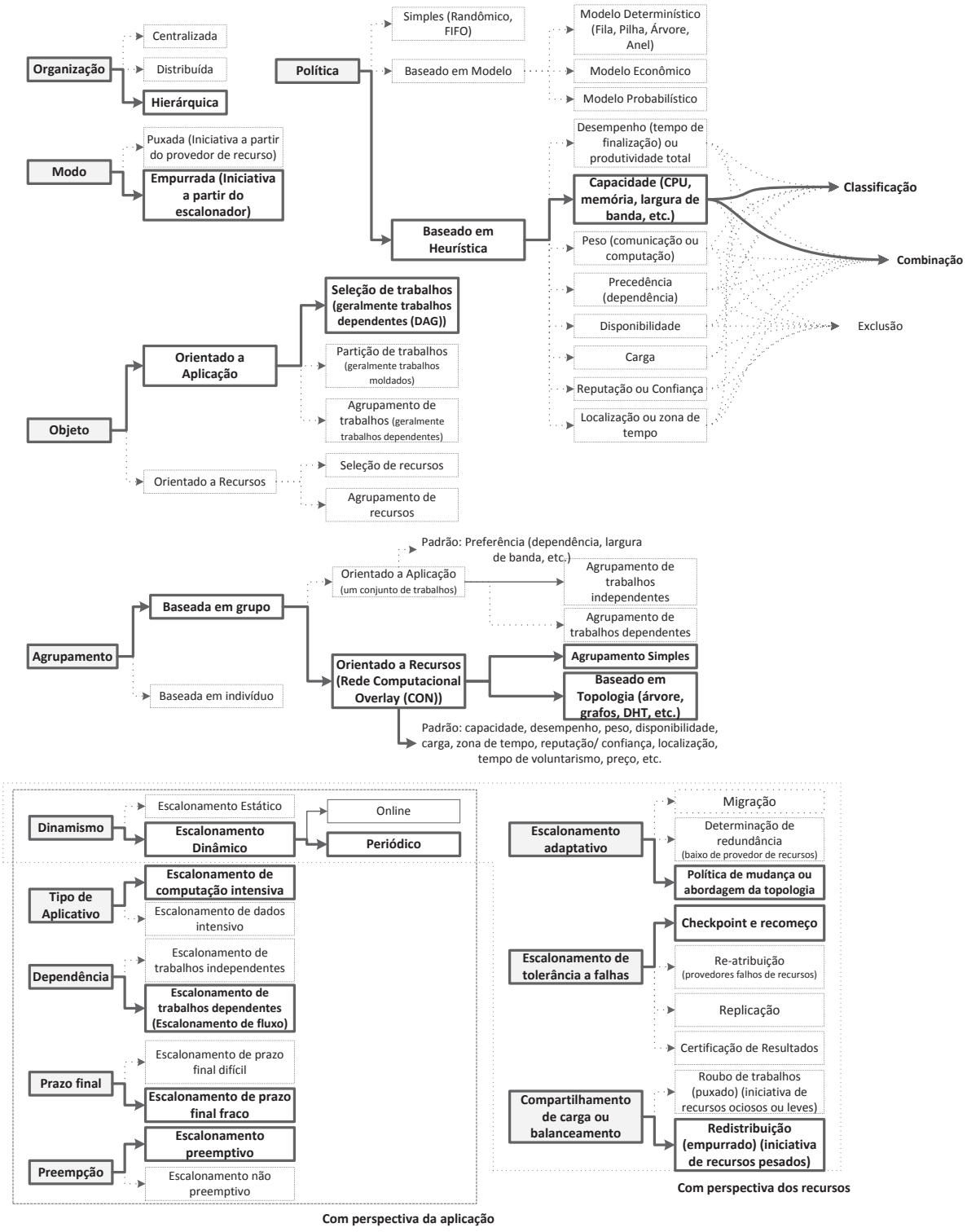
Elaborado pelo autor

4.2 Classificação Segundo a Taxonomia de Zhao, Liu e Li (2011) e Choi et al. (2006)

O modelo BSPonP2P foi criado para trabalhar em arquiteturas de Grade P2P, sendo uma combinação de Grades *Desktop* e P2P. Através das Taxonomias de Zhao, Liu e Li (2011) e Choi et al. (2006), será detalhado o BSPonP2P. Na primeira, a perspectiva é voltada ao escalonamento em uma visão de Grades *Desktop*. Porém, na segunda, o foco é voltado ao gerenciamento das unidades.

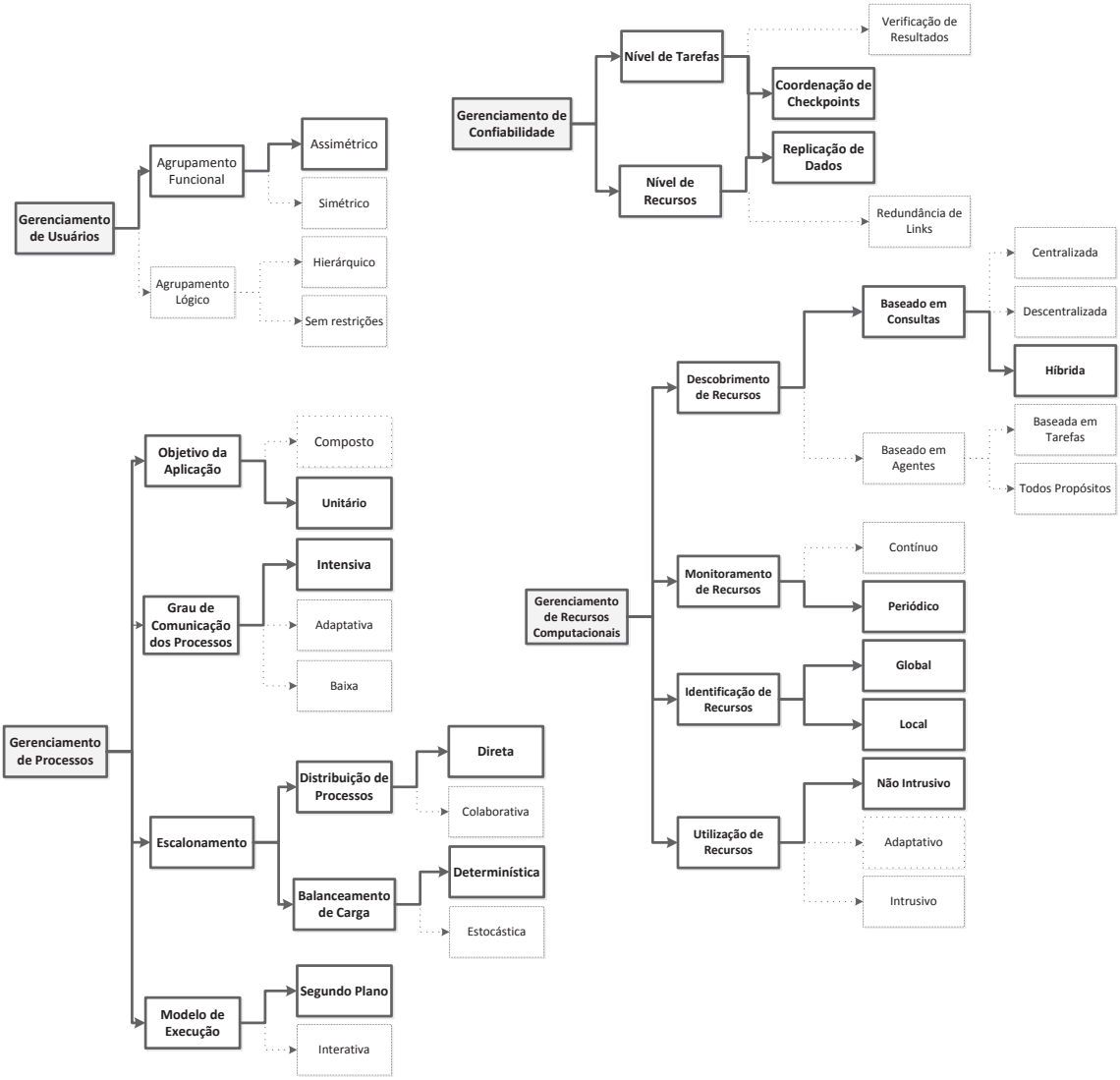
Nas Figuras 12 e 13, é apresentado o modelo conforme as taxonomias. No seguimento desta seção, elas são detalhadas conforme: (i) Arquitetura, (ii) Usuários, (iii) Tarefas, (iv) Escalonamento, (v) Recursos e (6) Segurança e Confiança.

Figura 12 – Taxonomia da Computação de Grades *Desktop*



Fonte: Elaborado pelo autor baseado em Choi et al. (2006)

Figura 13 – Taxonomia de Grades P2P

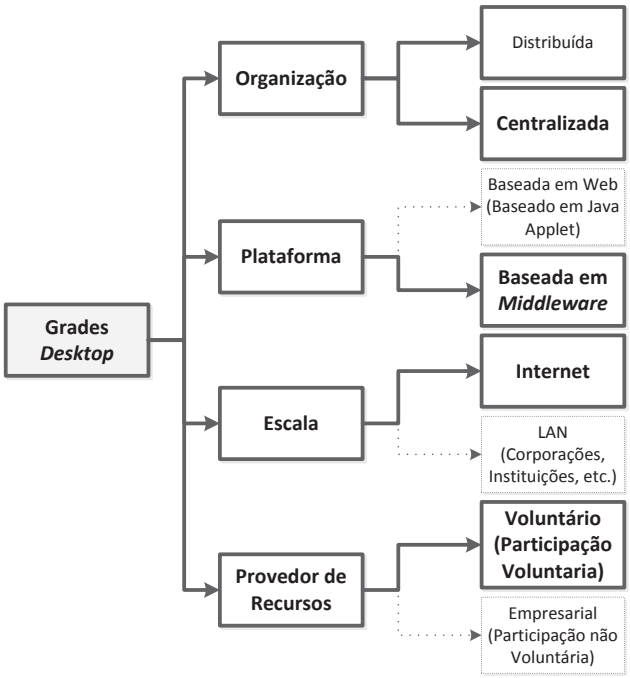


Fonte: Elaborado pelo autor baseado em Zhao, Liu e Li (2011)

4.2.1 Arquitetura

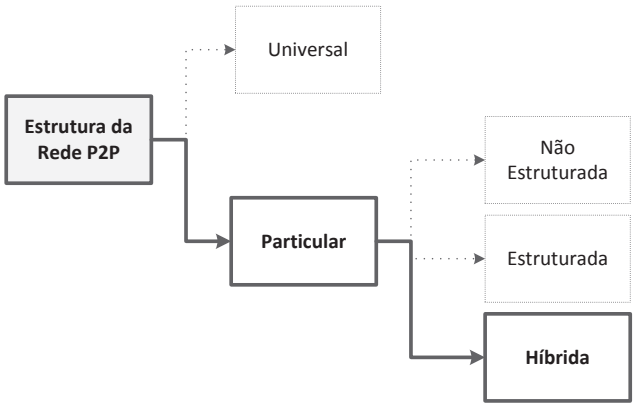
Nas Figuras 14 e 15, é demonstrado que o modelo utiliza uma organização híbrida (Centralizada e Distribuída). Isso acontece porque existem dois tipos de arquiteturas no BSPonP2P: a estruturada e a não estruturada. Na estruturada, a organização é feita de maneira distribuída (não há um servidor) pelo protocolo Chord. Na não estruturada, a organização é através de um servidor (Gestor de Grupo), e, sendo assim, ela é centralizada. Para controle dessa arquitetura é utilizado uma plataforma baseada em *middleware* que inicia um serviço de controle. O serviço tem a proposta de criar um ambiente voluntário para proporcionar recursos na escala da *Internet*.

Figura 14 – Arquitetura de Grades *Desktop*



Fonte: Elaborado pelo autor baseado em Choi et al. (2006)

Figura 15 – Arquitetura de P2P



Fonte: Elaborado pelo autor baseado em Zhao, Liu e Li (2011)

4.2.2 Gestão de Usuários

Gerenciamento de usuários em Grade P2P determina o modelo operacional do usuário final. A Figura 13 mostra a taxonomia para gerenciamento de usuários. Geralmente há duas maneiras de organizar os usuários em um sistema de Grade P2P: (i) Visão lógica; e (ii) Visão funcional.

O modelo BSPonP2P possui uma visão funcional porque utiliza o agrupamento assimétrico. Na sua estrutura existem dois papéis funcionais (Nó e Gestor de Grupo), onde cada um possui características diferentes.

4.2.3 Gestão de Processos

Gerenciamento de processos lida com questões relacionadas à submissão de trabalhos por usuários, incluindo uma caracterização do trabalho, a interação, a decomposição e o escalonamento. A taxonomia está resumida na Figura 13. O objetivo da aplicação pode ser unitário ou composto. O BSPonP2P tem o objetivo unitário. Ele utiliza um único modelo de programação paralela, que no caso é o BSP. Outra questão sobre a caracterização do processo é em relação ao grau de comunicação/computação. O modelo exige um intenso grau de comunicação, com comunicação entre os processos no BSP e também entre os participantes da CON para balanceamento de carga.

O balanceamento de carga requer monitoramento de estado dos recursos e migração dos processos em todo o substrato de rede. No modelo BSPonP2P a distribuição das processos é feita de maneira direta. O Gestor de Nó responsável pelo lançamento da aplicação faz o monitoramento e direciona diretamente o Nó que deve executar o processo. O balanceamento de carga determinístico usa serviços padrões de grade ou protocolos que utilizem DHT, como o Chord. A partir dessa premissa, como o modelo utiliza Chord na rede direcionada aos Gestores de Grupo ela é classificada como determinística. Nesse ambiente, os processos são executados em segundo plano, não havendo necessidade de interação com o usuário.

4.2.4 Escalonamento

A partir da Figura 12, é possível perceber que a taxonomia de escalonamento para Grades *Desktop*, conforme Choi et al. (2006), é definida por três perspectivas: aplicação, recursos e programação.

- **Perspectiva da Aplicação** - O ambiente do modelo utiliza BSP para trabalhar com processos dependentes, sendo que eles não possuem uma divisibilidade fixa. Além disso, a submissão dos processos não é determinística, não havendo o controle de quando será lançada.

- **Perspectiva dos Recursos** - No BSPonP2P, os provedores de recursos são autorizados a entrar e sair livremente, mesmo estando em meio a execuções. Neste tipo de caso, os participantes não são dedicados e eles atendem à CON e aos seus usuários primários. Nesse cenário, é evidente que há a necessidade de controle dinâmico da aplicação por haver constantes mudanças na carga dos provedores de recursos, oscilação da largura de banda e disponibilidade dos recursos.

Além disso, o ambiente é criado para atender à larga escala (no caso, a *Internet*), não se restringindo a um certo local. Sendo assim, não há padrão entre os recursos computacionais, criando um ambiente completamente heterogêneo;

- **Perspectiva do Escalonador** - O modelo BSPonP2P utiliza uma abordagem hierárquica para decisão de agendamento. O escalonamento de primeiro nível aloca diretamente os processos aos nós, enquanto o escalonamento de segundo nível os aloca para os Gestores de Grupo. Como o modelo inicia o trabalho a partir do escalonador, então ele é classificado como modo empurado, onde a política de escalonamento é baseada em heurística. Para o controle da política, o ingresso dos recursos na rede é baseado na sua entrada. Sendo assim, pode haver um agrupamento simples, caso a recusa seja direcionada à rede não estruturada ou baseada em topologia, caso receba o papel de Gestor de Grupo.

  Além disso, o sistema para adaptar-se às mudanças dinamicamente possui um escalonamento adaptivo. O controle sobre a consistência da execução é feito pelo serviço de *checkpoint* para, de certa forma, ser confiável nesse ambiente dinâmico. O compartilhamento de carga possui uma abordagem de redistribuição. Ela transfere (ou empurra) processos de nós levemente carregados para nós ociosos.

4.2.5 Gestão de Recursos

Gestão de recursos desempenha um papel crítico na CON de forma que abrange tanto a gestão de recursos de computação e da rede P2P, como indicado na Figura 13. Para os recursos de computação são estabelecidas quatro funções principais: descoberta, monitoramento, identificação e utilização de recursos.

A identificação dos recursos físicos é apresentada em unidades logicamente abstratas. No desenvolvimento do modelo, foi utilizado um espaço global de nomes para diferenciar as entidades virtuais, onde cada recurso é conhecido através de um identificador único. Para a localização dos recursos são utilizados diferentes métodos: um método para a rede não estruturada e outro para a estruturada.

O escalonamento para os recursos é uma decisão que depende do seu monitoramento. Com base na coleta dos contatos dos recursos, o monitoramento pode classificá-los em periódicos ou contínuos. O BSPonP2P faz monitoramento periódico. Regularmente os nós são contatados individualmente pelo Gestor de Grupo para coletar os dados (puxada) referentes a situação do equipamentos que estão sobre o seu dominio.

A descoberta de recursos foi categorizada em agentes e fundamentada em consultas. O BSPonP2P utiliza o mais comum, no caso, o método baseado em consulta. O modelo está constituído para trabalhar de maneira híbrida, com a arquitetura estruturada e não estruturada. Na estruturada, é usada a descoberta descentralizada promovida pelo Chord. Já a não estruturada é executada de maneira centralizada pelo Gestor de Grupo. Em relação à utilização dos recursos,

o modelo usa o não intrusivo de forma adaptativa. Ele avalia a utilização dos recursos para a contribuição de ciclos computacionais.

4.2.6 Gestão de Tolerância à Falhas

O ambiente de Grades P2P estende o ambiente de Grades *Desktop* tradicional à escala global. No entanto, essa extensão também apresenta recursos não confiáveis e voláteis. Por isso, manter a confiabilidade é de extrema importância, envolvendo todos os níveis verticais da arquitetura. A confiabilidade é dirigida principalmente aos aspectos de desenho do sistema para além dos usuários. Nessa parte, categorizar tecnologias de confiabilidade é aplicado a vários níveis em um sistema típico. A taxonomia está resumida na Figura 13 de forma que é percebido que o modelo utiliza alguns recursos para controle do ambiente. A coordenação de *Checkpoints* e a replicação dos dados gerados por eles criam ferramentas para a gestão da confiabilidade.

4.3 Modelo de Máquina para Processamento Distribuído

Nesta seção, são apresentadas algumas definições do modelo BSPonP2P, tais como: (i) Definição de atores e termos - padroniza os termos que são utilizados na definição do modelo; (ii) Arquitetura - apresenta a arquitetura utilizada no BSPonP2P; e (iii) Roteamento - informa como é feito o roteamento dentro do modelo.

4.3.1 Definição de Atores e Terminologia

Segue-se uma breve explicação das terminologias e dos atores utilizadas no modelo:

- **Rede Computacional *Overlay* (CON)** - A Rede Computacional *Overlay* (CON) (RAJ; HARIKUMAR, 2014) é o ambiente virtual criado para a estruturação da rede. Esta rede é criada e estruturada em tempo de execução, buscando atender às necessidades existentes no ambiente (escalonamento e reescalonamento). A entrada dos participantes na CON influencia o seu papel na rede;

- **Nós *(Peers)*** - O recurso computacional que recebe o título de Nó é aquele que está ligado ao Gestor de Grupo. Sendo assim, o título é dado para aquele que executa ou encaminha os trabalhos a serem realizados, aquele que cria a necessidade;

- **Processo** - O Processo é a função executada pelo *host* ao entrar na CON;

- **Tarefas** - A tarefa é demoninada como t, sendo que é a menor unidade de execução no modelo. Os processos direcionam as tarefas a serem executadas;

- **Gestores de Grupo (*Brokers*)** - Os Gestores de Grupos são responsáveis pelo gerenciamento dos grupos de nós que são definidos como b. As principais responsabilidades dos

Brokers são: (i) servir como gestores da rede de nós; (ii) efetuar o escalonamento dos processos; (iii) controlar a execução dos processos; (iv) controlar os *checkpoints* (JAIN; CHAUDHARY, 2014); e (v) gerenciar as redundâncias de dados;

Juntamente com a definição dos papéis dos atores, seguem algumas definições da rede do modelo:

- **Rede de Nós/Grupo de Nós (Rede Não Estruturada)** - É o grupo criado a partir de um conjunto de nós que estão interligados formando uma rede. Por definição, denomina-se P, onde $P = \{p_0, p_1, p_2, p_3, ..., p_n\}$. Esta, por sua vez, será criada de maneira não estruturada e gerenciada de forma centralizada. Os nós estarão ligados ao Gestor de Grupo.

- **Rede de Gestores/ Anel de *Brokers* (Rede Estruturada)** - A Rede de Gestores é formada pela interligação dos Gestores de Grupo, que formam um grupo. No modelo, B denomina-se a Rede de Gestores, onde $B = \{b_0, b_1, b_3, ..., b_n\}$. Esta rede é criada e gerenciada a partir do protocolo Chord, e, sendo assim, ela é classificada como rede estruturada.

4.3.2 Estrutura de Rede

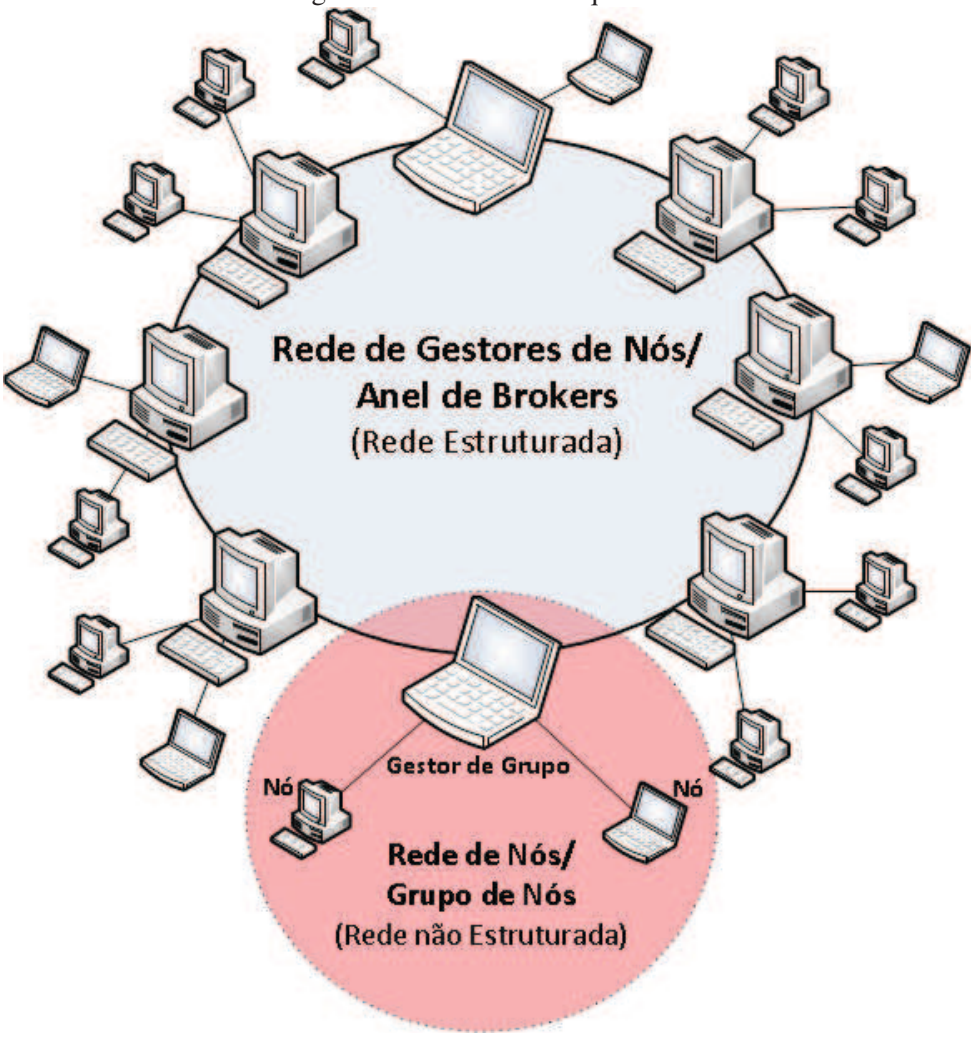
O modelo BSPonP2P utiliza alguns recursos vindos da arquitetura P2P, tais como as definições de arquitetura estuturada e não estruturada, confome Figura 16. No caso, a arquitetura estruturada foi definida por trabalhar com DHT, a qual faz boa gestão na troca de mensagens e roteamento. Ela foi utilizada na Rede de Gestores a fim de que esta seja uma rede de alta escala. A rede desestruturada foi definida por possuir bons recursos na criação de redes com necessidades específicas. No caso, é necessário que haja comunicação e troca de informações detalhadas entre o nó e seu gestor.

A entrada de um recurso computacional na rede é dada a partir de um endereço IP conhecido, o qual o direciona para um dos grupos. Apresenta-se, a seguir, a descrição de como é criada cada uma das redes no modelo:

- **Criação da Rede Não Estruturada** - A criação da rede não estruturada é gerada a partir da entrada dos participantes na CON. Esta rede é formada pela interligação de n nós, sendo n o limite de participantes do grupo. Ao alcançar $n + 1$ nós na rede, o *middleware* abre outro grupo para que os novos participantes entrem. O limite influencia a definição do papel do nó na rede, podendo ser: (ii) Gestor de Grupo, quando o participante for o primeiro a entrar na rede; (ii) Nó, quando já houver um Gestor de Grupo e a quantidade de participantes for menor que o limite estipulado.

  Ao entrar na rede, o participante deve informar o quanto de seu recurso computacional irá disponibilizar, podendo ser: (i) Total, quando não estiver em uso o equipamento, o *middleware* contará com a quantidade total de recursos; (ii) Percentual indicado, quando

Figura 16 – Modelo da arquitetura



Fonte: Elaborado pelo autor

o usuário define quanto ele deseja disponibilizar do recurso. Dentro do *middleware* instalado no computador do participante, é iniciada uma máquina virtual, a qual é denominada com Nó Virtual (NV) (MA; HE; MA, 2014). Nesse NV, o recurso computacional é disponibilizado conforme a seleção do usuário.

Para manter atualizados os dados do Nó, periodicamente, em intervalos de tm segundos, o Nó encaminha as suas informações para o seu Gestor de Grupo. A constante tm é definida pelo Gestor de Grupo. Por definição, caso não haja o recebimento das informações o modelo aguardará mais dois recebimentos e caso não haja o Nó é considerado como inativo e desconectado da rede.

Cada grupo possuirá um número de identificação. Sendo assim, se algum Nó abandonar um dos grupos, o próximo participante a entrar na CON será direcionado pelo *middleware* para um dos grupos com disponibilidade de participação. O *middleware* fará uma classificação crescente dos IDs de grupos e sempre preencherá primeiramente os grupos com o menor ID.

- **Criação da Rede Estruturada** - A rede estruturada é definida para a interligação entre os Gestores de Grupo. Ela é iniciada a partir da existência de, no mínimo, dois gestores de grupo na CON. O protocolo que é utilizado para a estruturação da rede é o Chord, que usa uma forma eficiente de gerenciamento da DHT.

  Nesse ambiente criado, os gestores de grupo trocarão informações sintetizadas dos seus grupos, tratando estes como *clusters*. As informações conterão os totais do que o grupo está disponibilizando para a CON. Essas informações serão armazenadas na *Finger Table* (LIN et al., 2014). Essa tabela é um recurso disponibilizado pelo protocolo Chord.

4.3.3 Gerenciamento do Roteamento de Mensagens

Como o ambiente é algo dinâmico, muitos participantes podem abandonar a rede de forma inesperada, e, sendo assim, o *middleware* deve reorganizar a rede e tornar a mudança imperceptível para todos os outros participantes. Serão tratadas duas saídas no modelo: (i) saída do Gestor da Rede; e (ii) saída do Nó.

As informações que o Gestor do Nó reúne são divididas e armazenadas no próprio grupo ou em outro grupo. Quando um Gestor de Grupo sair inesperadamente da rede, um Nó do mesmo grupo assumirá o seu posto. O Nó que receberá esse posto será aquele que estiver, há mais tempo, ligado na rede, e a seleção será feita dessa maneira, acreditando que é um participante confiável por permanecer mais tempo conectado. Ao receber o papel de Gestor de Grupo, ele será atualizado com as informações relevantes, as execuções atuais e os resultados já obtidos.

O Nó, ao abandonar a rede, estiver executando um processo e não o tiver terminado, o Gestor possuirá informações parciais dessa execução e, sendo assim, selecionará novamente outro participante conforme o resultado do escalonamento (função que será explicada nas próximas seções). A execução continuará conforme o ponto que estava armazenado no Gestor de Nós.

4.4 Modelo de Aplicação

O modelo de aplicação é baseado em aplicações BSP. Entretanto, o modelo aplicado no BSPonP2P diverge do modelo tradicional do BSP porque o tradicional foi desenvolvido para recursos homogêneos em *Cluster*. Para trabalhar com a heterogeneidade presente no ambiente Grade P2P foram aplicadas migrações. Sendo assim, o modelo tradicional do BSP foi estendido de forma que uma aplicação feita para execução em Grade P2P pode ser executada em *Cluster*. Para execução em *Cluster* não há necessidade de mudanças, visto que no modelo apenas foi incrementado o recurso de controle de entradas e saídas.

A comunicação dentro da aplicação é dividida em grupos para agilizar o processso, sendo um direcionado aos Gestores de Grupo e outro aos Nós. Além disso, o modelo possui três divisões para o direcionamento dos processos: (i) Escalonamento de primeiro nível, definição do grupo que é o primeiro após o lançamento para execução; (ii) Escalonamento de segundo nível,

definição de qual nó fará a execução; (iii) Reescalonamento, ponto mais importante da estratégia, pois, a partir de dados das execuções, reavalia o ambiente. Como base do reescalonamento, é utilizado o trabalho Rosa Righi et al. (2011).

4.4.1 Comunicação

A comunicação dentro da CON trabalhará em dois níveis: (i) Primeiro nível - comunicação entre o Grupo de Gestores; e (ii) Segundo nível - comunicação entre o Grupo de Nós. A comunicação foi dividida para tratar os dois ambientes criados no modelo: a rede estruturada, que irá trabalhar com DHT, e a rede não estruturada, a qual irá trabalhar de maneira centralizada.

- **Primeiro Nível - Grupo de Gestores:** A comunicação no Grupo dos Gestores é controlada a partir do protocolo Chord, o qual distribui as informações de roteamento entre os participantes e trabalha esse roteamento de maneira descentralizada. O protocolo necessita apenas de informações de poucos participantes para conseguir estruturar todo o roteamento. Com isso, são gerenciadas as informações de entradas e saídas de participantes na CON.

  Quando um novo Gestor de Grupo entrar na rede, ele receberá a *Finger Table* para assim participar da rede e conseguir montar o roteamento da troca de mensagens. Esse recurso é disponibilizado pelo protocolo Chord. Além disso, os Gestores informarão aos outros Gestores, aos quais estão ligados, quando houver mudanças nos outros Gestores que estão ligados a eles na *Finger Table* (entradas ou saídas).

- **Segundo Nível - Grupo de Nós:** A comunicação neste nível é intensa porque o ambiente é dinâmico e, assim, o modelo exige constantes verificações para averiguar a permanência dos participantes na rede. Em determinados períodos (conforme definido anteriormente), a rede não estruturada troca mensagens a fim de atualizar a situação do nó na rede.

  Além disso, o Nó, ao entrar na rede, envia uma mensagem ao Gestor do Grupo, informando os dados para que sejam utilizados no escalonamento(informações, as quais serão indicadas na seção de Escalonamento). Na primeira comunicação, o Gestor de Grupo cria um ID para o usuário e armazena os dados referentes ao participante em sua memória. O ID é um número sequencial que o Gestor de Grupo fará a gestão.

  As execuções dos processos são gerenciadas pelo Gestor de Grupo, que envia a tarefa e os dados para serem executados pelo Nó, que, conforme já descrito, controla a permanência do participante na rede. O Nó, por sua vez, está em constante comunicação com os outros executores do Grupo de Tarefas porque o modelo BSP exige tal comunicação. Para facilitar a comunicação, a CON é moldada (reestruturada) para gerir esse ambiente. Sendo assim, todos os executores (Nós) são ligados ao Gestor de Grupo em que surgiu a necessidade de execução, sendo essa estratégia utilizada para diminuir a sobrecarga de comunicação. Dessa forma, a distância percorrida pela mensagem é menor.

Todo o controle de troca de mensagens é feito pelo módulo MOM (*Message Oriented Middleware*), recurso disponível em todos participantes (Gestores de Grupo e Nós). As mensagens são lidas conforme sua ordem de chegada. Sendo assim, é utilizado o método Primeiro que Entra Primeiro que Sai (PEPS).

4.4.2 Lançamento de Trabalhos na Aplicação

O lançamento de trabalhos na aplicação é dado por solicitação dos participantes (nós), conforme Figura 17. Na Etapa 1, o nó solicitante encaminha o trabalho para o seu Gestor de Grupo (*Broker*). O Gestor de Grupo, por sua vez, entra na etapa de escalonamento. Na Etapa 2, é feito o escalonamento de maneira simples: os gestores de grupo são ordenados pela sua capacidade. Os que possuírem maior capacidade receberão inicialmente os primeiros processos.

Na Etapa 3, o Gestor de Grupo que está solicitando a execução cria um canal direto com os nós que participarão da execução, conforme definido na Etapa 2. Na Etapa 4, os processos são encaminhados aos nós e é iniciada a aplicação BSP. A Etapa 5 caracteriza o reescalonamento, que é dado em dois níveis:

- **Primeiro Nível:** A avaliação de primeiro nível indicará qual grupo de nós executará determinada tarefa. O Gestor de Grupo coletará as informações relevantes para o cálculo da Equação 3.4. A Equação é baseada no modelo MigBSP(ROSA RIGHI et al., 2011). Porém a diferença encontrada entre os modelos BSPonP2P e MigBSP é o tipo de rede no qual foram aplicados, pois o BSPonP2P usa Grades P2P e o MigBSP utiliza Grades de Computação.

  Ao tratar de ambientes dinâmicos, o BSPonP2P utiliza informações dos usuários e equipamentos para o cálculo da Equação 3.4. Os dados referentes às informações usadas são baseados em uma média aritmética calculada a partir da Equação 4.1 e utilizada na Equação 4.2, que trata da computação na equação de PM. $\overline{X}_{Equipamento}$ e $\overline{X}_{Usurio}$ utilizando a média de disponibilidade dividida pela capacidade que possuem. A capacidade é dada pela quantidade de Flops dísponíveis pelo recurso computacional. A amostra do cálculo usa no mínimo, três avaliações e, no máximo, dez. Ela se baseia nos últimos registros recebidos. Caso não haja três avaliações, o modelo define o seu resultado como um (totalmente disponível).

$$\overline{X}_{Dispo} = \frac{\overline{X}_{Equipamento} + \overline{X}_{Usurio}}{2} \tag{4.1}$$

$$Comp(i,j) = \overline{X}_{Dispo}.P_{Comp}(i).CTP(i).ISet(j) \tag{4.2}$$

Segue uma breve avaliação da utilização das características:

  - **Usuário**: Esta característica é utilizada afim de encontrar uma razão na utilização do equipamento. A partir do usuário que está logado no equipamento, é buscado um padrão de utilização simples, que é dado pela média aritmética de utilização pelo usuário ($\overline{X}_{Usurio}$). Isso é relevante devido a cada usuário possuir necessidade distintas, sendo assim, a necessidade de um usuário pode atingir 80% de utilização do recurso computacional, porém outro usa somente 5% do recurso. A intenção da utilização dessa métrica é que, mesmo o equipamento estando disponível e seja um usuário que exija recurso computacional, a prioridade deve ser daquele que não exija.
  - **Equipamento**: A carcaterística equipamento é inclusa afim de encontrar uma razão na sua utilização porque vários usuários podem logar nele. Mesmo tendo a média de utilização do usuário, é desejado que o equipamento que tenha maior disponibilidade e capacidade seja utilizado. No caso, o equipamento pode estar com um usuário que exija computação, porém, ao avaliar algumas amostras, percebe-se que a média de utilização é baixa. Isso acontece porque houve constante troca entre os usuários logados. Essa métrica vem para ponderar, de maneira positiva, esse tipo de cenário.

- **Segundo Nível:** A avaliação de segundo nível é utilizada para definir qual nó dentro de um grupo fará a execução de uma tarefa. A definição do nó executante é feita a parir da avaliação de disponibilidade do equipamento. Nesse ponto, é feita uma avaliação, onde é utilizada uma amostra de, no mínimo, três avaliações e, no máximo, dez avaliações de disponibilidade (quantidade de recurso computacional disponível). A amostra é fundamentada nos últimos registros recebidos no Gestor de Grupo. Não havendo três avaliações, o modelo define o seu resultado como totalmente disponível (capacidade computacional disponibilizada pelo equipamento).

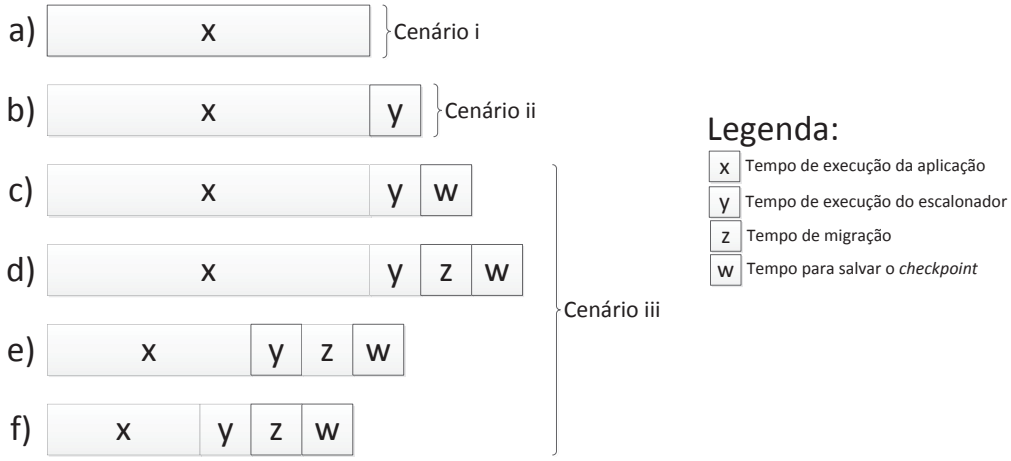
O reescalonamento é feito nas barreiras das *supersteps* que são definidas por α. O índice de reescalonamento segue as regras do Primeiro Nível, porém com os dados atualizados dos grupos e nós. Nesta etapa, são caracterizadas as migrações de processos. A Etapa 5 é repetida conforme o número de *supersteps* e os valores de *alpha* encontrados. Na Etapa 6 é encontrado o fim da execução, em que são encaminhados os resultados para o nó solicitante pelo Gestor de Grupo.

4.5 Serviços de Controle do Modelo

O diferencial do modelo é destacado pela migração e pelo *checkpoint*. Na Figura 18, são apresentados três cenários e os resultados esperados. O cenário i (parte "a") é a execução da aplicação sem os tempos de migração, escalonador e salvamento do *checkpoint*. O cenário ii (parte "b") é a execução da aplicação e do escalonador. Já o cenário iii (partes "c", "d", "e"e

"f") é a execução do modelo com todos os serviços (migração e *checkpoint*). Nas partes "c", "d", "e"e "f"da Figura 18 são demonstrados os resultados esperados. Na parte "c", o modelo constatará que não há como efetuar migrações, porém manterá a consistência com o *checkpoint*. Na parte "d", todos os serviços estarão ativos. Na parte "e", os serviços estarão ativos e, com as migrações, o tempo será menor que a parte "b". Nesse cenário, o modelo demonstrará o desempenho que as migrações incorporaram. A parte "f"é o melhor cenário possível, onde todos os serviços estarão ativos e o modelo identificará os melhores recursos computacionais para execução. Nesse cenário, o desempenho será melhor que a parte "a", que é a simples execução da aplicação.

Figura 18 – Resultados previstos



Fonte: Elaborado pelo autor

Nessa seção, são apresentados os serviços de controle presentes na *superstep* do BSP. Esses serviços foram implementados a fim de garantir o controle de falhas, além de otimizar a execução das tarefas.

4.5.1 *Checkpoint*

O modelo faz uso do ambiente Grades P2P, o qual é dinâmico e possui constantes entradas e saídas de nós da rede. Nessas saídas, os nós poderão estar executando processos, e, dessa maneira, a execução terminará com problemas porque uma parte não será entregue. Pensando nesse problema e aproveitando as paradas para reavaliação na *superstep*, o BSPonP2P fará uma foto do ambiente (*snapshot*/*checkpoint*) para conseguir recuperar a execução perdida. Nessa foto, estarão salvas as informações referentes aos processos e dados da última *superstep*.

As informações do *checkpoint* serão enviadas ao Gestor do Grupo e ele as salvará na *Finger Table*. Após salvas as informações, o protocolo Chord irá distribuí-las entre os Gestores de Grupo e gerará redundância de dados. Desse modo, caso um participante deixe a rede no meio

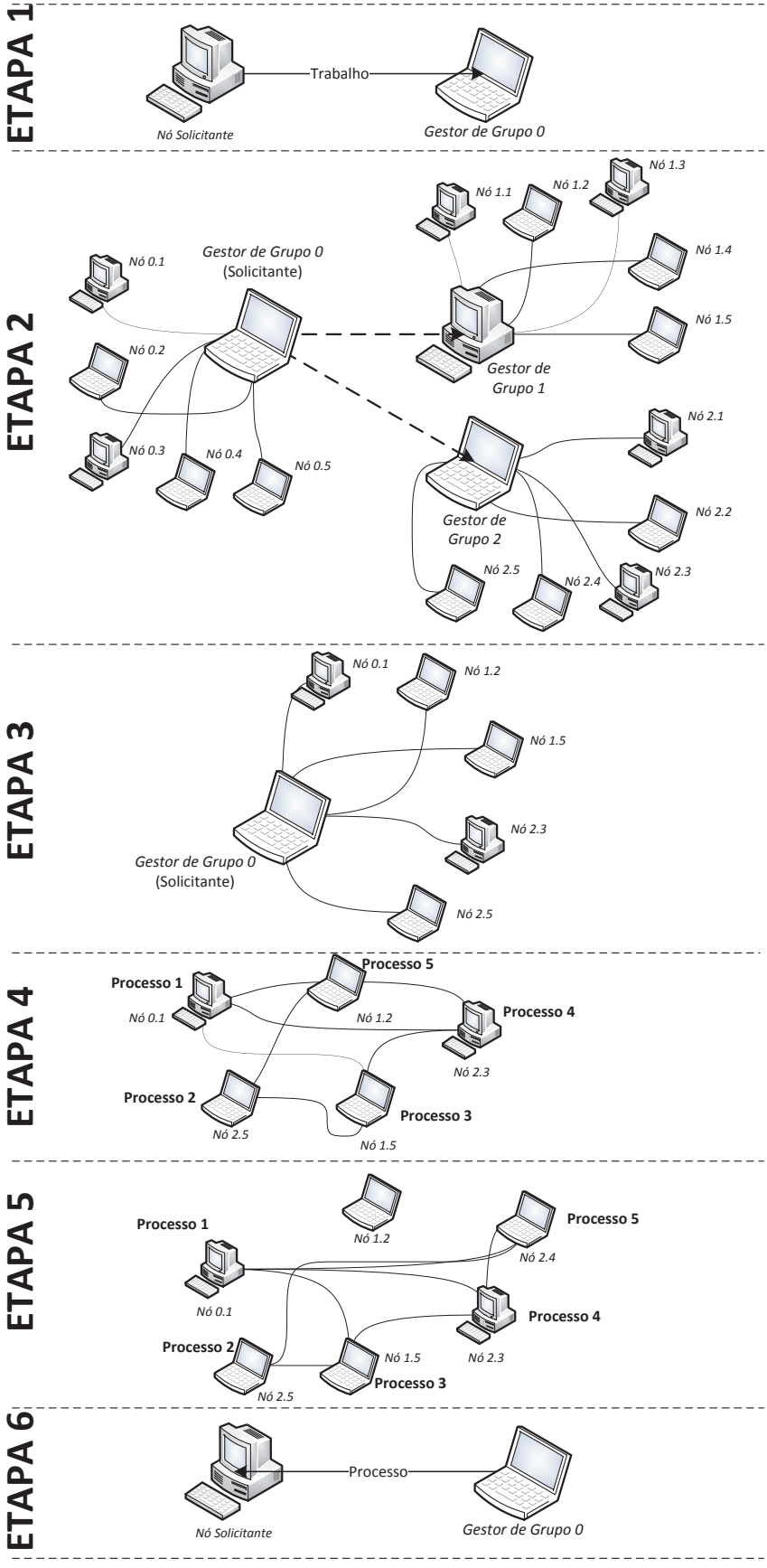
de uma execução, o Gestor de Grupo deverá identificar o último *checkpoint* salvo e iniciá-lo novamente. A identificação da falha é dada a partir da não resposta do nó ao Gestor do Grupo nas avaliações periódicas. Esse processo é extremamente necessário porque o BSP é uma aplicação fortemente acoplada.

4.5.2 Migração de Processos

Ao tratar de compartilhamento de recursos computacionais é necessário que haja controle nas execuções dos processos. Por isso, nas barreiras de reavaliação (no reescalonamento), de acordo com o valor obtido por PM, a migração dos processos de um nó para o outro será conduzida pelo Gestor de Grupo. O valor de PM para que haja migrações poderá ser influenciado por:

- **Reavaliação**: Nas barreiras onde houver reescalonamento, o escalonador avaliará o ambiente e verifica os processos ativos. Sendo assim, ele as mantém ou destina a outro equipamento;
- **Necessidades dos usuários**: Por definição, o modelo BSPonP2P sempre deixará disponível a quantidade de recursos solicitada pelo usuário.

Figura 17 – Lançamento de trabalhos. Etapa 1 - solicitação da execução do trabalho. Etapa 2 - gestor avalia os recursos disponíveis. Etapa 3 - criação da CON com os recursos selecionados. Etapa 4 - alocação dos processos para os recursos. Etapa 5 - reavaliação do ambiente e migrações. Etapa 6 - os resultados são entreges ao solicitante.



Fonte: Elaborado pelo autor

5 IMPLEMENTAÇÃO

A implementação é baseada em simulação porque esse tipo agiliza a implementação e facilita a criação do ambiente, processos e diretivas de execução. Além disso, permite o aumento de escala do ambiente de maneira fácil.

Nesse capítulo, serão demonstrados pontos referentes à implementação do modelo BSPonP2P, estando dividido da seguinte maneira: Seção 5.1 - apresenta a definição do simulador utilizado; Seção 5.2 - aborda a estruturação da plataforma e o desenvolvimento dos processos; e a Seção 5.3 - demonstra a construção do modelo.

5.1 Definição do Simulador

Para a definição do simulador que foi utilizado no modelo BSPonP2P, algumas perspectivas foram analisadas, como:

- Criação de processos: Os processos devem ser executados em uma região específica do código ou função;
- Troca de mensagens baseadas no desenvolvimento da aplicação: O simulador deve permitir a troca de mensagens entre todos os participantes em qualquer processo da aplicação. Além disso, deve permitir comunicação assícrona;
- Migração de processos: Possuir um mecanismo para a realocação dos processos para outro recurso disponível;
- Diretivas de execução: O simulador deve permitir capturar o tempo corrente de simulação, o número de instruções e também a troca em *bytes* na comunicação.

Na barreira de sincronização de uma *superstep* BSP deve ser possível implementar diretivas simples de comunicação *send-receive*. A partir da descrição dos requisitos necessários, foram verificados dois simuladores: (i) Peersim (MONTRESOR; JELASITY, 2009); e (ii) Simgrid (CASANOVA; LEGRAND; QUINSON, 2008). Esses dois simuladores oferecem a construção de ambientes heterogêneos garantindo a velocidade de comunicação na rede e processos.

O PeerSim cria ambientes extremamente escaláveis que suportam cenários dinâmicos. O Simgrid objetiva facilitar a área de pesquisa, envolvendo aplicações distribuídas e paralelas de escalanomento em plataformas distribuídas. O último simulador foi escolhido para a implementação do modelo devido a preencher todos os requisitos e ser largamente utilizado na comunidade científica.

No simulador foi utilizada a *interface* MSG. Ela permite o estudo de aplicações baseadas em processos sequenciais concorrentes. A MSG oferece uma API para gerenciamento dos processos, nós e tarefas. Além disso, possui funções para o gerenciamento do sistema operacional.

As funções principais da última área permitem tratar as execuções das tarefas, pausas dos processos, capturas do tempo dos envios e recebimentos das tarefas.

5.2 Plataforma de Execução e Mapeamento Inicial dos Processos

O Simgrid permite especificar dois arquivos a partir da linguagem XML, que são:

- Plataforma de computação: Neste arquivo ficam as seguintes informações: (i) dados do *host*, que apresenta o nome, poder computacional, quantidade de processadores e disponibilidade computacional; (ii) dados de rede, que contemplam os dados referentes às rotas de roteamento, à largura de banda, às conexões entre os *hosts* e à latência.

- Mapeamento inicial dos processos: Permite atribuir características e funções aos processos criados na aplicação.

Na arquitetura do Simgrid, primeiramente ao iniciar a aplicação os processos são mapeados em *hosts*. Nesse processo, é atribuída uma função ao *host* conforme o arquivo de desenvolvimento. Porém, a definição do processo é mudada pelo BSPonP2P em tempo de execução. Isso acontece porque a função está ligada ao controle de entradas na rede.

5.3 Implementação da Simulação do BSPonP2P

O mecanismo de barreira foi implementado utilizando um simples algoritmo centralizado. O menor processo dos nós (o qual não pode ser gestor) recebe as mensagens de todos os outros processos. Quando ele recebe a última mensagem, sinaliza a todos que podem iniciar a computação da seguinte *superstep*. Cada processo captura o tempo de sua *superstep* na função de barreira. Ele grava esse tempo em um vetor local e o passará para o seu gestor de grupo quando o reescalonamento for ativado. Na função de barreira, são atualizadas as métricas de Comunicação, Computação e Memória. Além disso, na mesma função é controlado o $alpha'$ e a tentativa da chamada de escalonamento dos processos.

Foram desenvolvidos controles para capturar os tempos de computação e comunicação. Eles atuam sobre as funções `MSG_task_execute`() e `MSG_task_get_from`(). No início de cada função, é capturado o número de instruções ou *byte* e o tempo inicial. No final de cada uma delas, é pego o tempo para avaliar o tempo de execução.

Os Gestores de Grupos, os quais foram definidos pelo BSPonP2P, executarão a função `broker_function`() e armazenarão uma lista com os nós sob sua jurisdição. Os nós que estiverem sob a sua jurisdição receberão a função `node_function`(). Os Gestores de Grupo alcançam os processos, variando sua responsabilidade e verificam os processos BSP que estão em execução. A chamada de reescalonamento faz retornar um ou mais processos candidatos, dependendo da heurística implementada. O procedimento de migração é executado com a combinação da função `MSG_process_change_host`(). A execução é seguida por um atraso

igual ao valor da métrica de Memória $Mem(i, j)$, considerando o processo i no conjunto j. A função `MSG_process_sleep`() é usada para oferece facilidade de atrasos.

6 RESULTADOS

Neste capítulo, serão demonstrados pontos referentes aos resultados da implementação do modelo BSPonP2P. Está dividido da seguinte maneira: Seção 6.1 - apresenta a definição da metodologia de como o modelo foi avaliado; Seção 6.2 - oferece os resultados obtidos com o modelo; e a Seção 6.3 - expõe os benefícios do modelo, avaliando os resultados.

6.1 Metodologia

Esta seção apresenta algumas decisões relacionadas às avaliações do modelo BSPonP2P. Primeiramente, serão discutidos em detalhes os cenários para testar o BSPonP2P. A chave deste tópico é demonstar as diferentes avaliações das execuções e verificar o comportamento do modelo. Em segundo lugar, será exposta a infreestrutura que foi assumida para os testes. E, no final, será abordada a aplicação utilizada para avaliar o modelo.

6.1.1 Cenários de testes

O modelo BSPonP2P em relação ao desempenho foi avaliado em três cenários: (i) A simples execução da aplicação; (ii) A execução do escalonador do BSPonP2P sem migração e com *checkpoint*; e (iii) A execução do escalonador do BSPonP2P com migração e com *checkpoint*. A Tabela 5 sumariza os cenários citados. A ideia é demonstrar o comportamento normal do BSPonP2P(cenário iii) e a situação onde todas as migrações são inviáveis (cenário ii). No cenário ii, é medida a sobrecarga relacionada ao escalonamento computacional e o custo da troca de mensagens entre o Gestor de Grupo e os nós. O cenário ii consiste em executar todos os cálculos de escalonamento e todas as decisões sobre qual processo realmente migrar, mas não executar as migrações. No cenário iii adicionam-se os custos das execuções das migrações.

Tabela 5 – Cenários de testes

Aparelho de Teste	**Execução da Aplicação**	**Execução BSPonP2P**	**Ativação da Migração**
Cenário i	*		
Cenário ii	*	*	
Cenário iii	*	*	*

Fonte: Elaborado pelo autor

A comparação entre os cenários i e ii refere-se à sobrecarga imposta pelo BSPonP2P. O cenário ii sempre apresenta tempo maior em relação ao tempo de i porque, além da aplicação, estará executando a função do BSPonP2P. A análise dos cenários i e iii demonstrará o ganho ou perda de desempenho quando usadas migrações. É possível afirmar que a realocação de um ou mais processos do BSP pode superar os custos de migração e escalonamento. A verificação dos cenários ii e iii é pertinente para observar a modificação de tempo de uma situação que tem sobrecarga para outra onde migrações possam ocorrer.

A validação em relação à recuperação também foi confirmada. Nesse cenário, foi avaliado o tempo de retomada da execução com o *checkpoint* e comparado com o tempo sem esse serviço, que é dado através da execução total, em que, havendo a falha, o sistema retoma o início da execução. Já o tempo com o serviço ativo ele retoma a partir da última barreira salva. A comparação vai ser baseada em uma saída inesperada da rede. A saída é de um *host* que esteja executando um processo no BSPonP2P. No cenário, foi avaliado a saída do *host* em uma determinada *superstep*.

6.1.2 Ambiente de Teste na Arquitetura de Grades P2P

Tecnologias envolvendo Grades P2P são bastante populares. Isso ocorre porque utilizam computação voluntária, que aproveita a ociosidade de recursos computacionais. Considerando isso, foi modelada uma infraestrutura com três gestores de grupos e dois nós ligados a cada gestor. A infraestrutura é heterogênea em termos de capacidade de processamento, assim como capacidade de largura de banda.

No ambiente criado, somente um processador dos nós é oferecido para execução das aplicações BSP. A ideia básica é preencher um grupo e então passar para outro. Foi mapeado um processo para cada nó, objetivando que cada um tenha um único processador. Caso a quantidade de processos seja maior que de processadores, o mapeamento começa com o primeiro conjunto. Isso acontece quando mais de 8 processos forem mapeados na arquitetura.

O BSPonP2P será testado em cima da plataforma (Figura 19) Grid5000[1] (CAPPELLO et al., 2005), que servirá como rede física. Esta plataforma é divulgada e constantemente utilizada no meio científico (YILDIZ et al., 2014). Com este parecer é acreditado que ao mapear a CON em cima desta rede, os resultados buscarão ser os mais próximos da realidade.

O mapeamento da *overlay* é dado a partir da entrada dos *hosts*. O Gestor de Grupo trabalhará com $n = 10$, isto significa que ligado ao Gestor de Grupo haverão 10 *hosts*. Os processos serão lançados pelos primeiros *hosts* que entrarem na CON, os quais receberam o papel de Nó. Cada *host* receberá somente um processo.

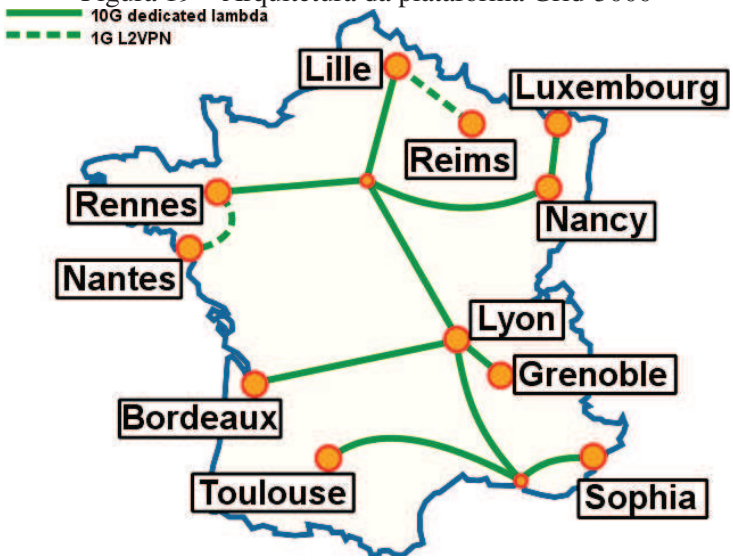
6.1.3 Aplicação para Avaliação

O método Lattice Boltzmann (SCHEPKE; MAILLARD, 2007) é baseado na equação Boltzmann, que utiliza computação discreta. Na equação, é considerado um típico volume de elementos de fluidos para compor uma coleção de partículas que são representadas por uma função de distribuição de velocidade para cada fluido que compõe cada ponto da grade. Por isso, é uma forte técnica para modelar uma grande variedade de complexos problemas de fluxo de fluidos.

A implementação foi modelada através da divisão do volume de dados, conforme pode ser visto na Figura 20. Os blocos serão divididos continuamente pelos processos. Múltiplas cópias

[1] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

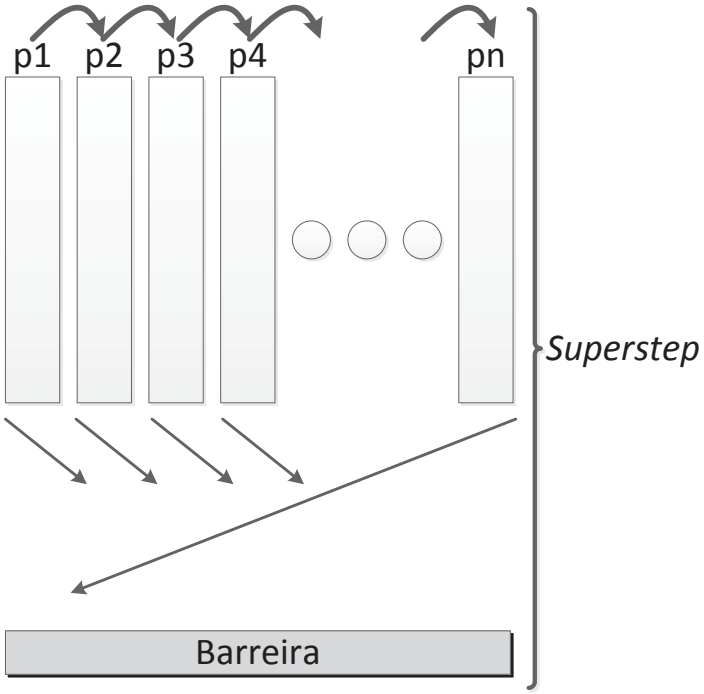
Figura 19 – Arquitetura da plataforma Grid'5000



Fonte: (CAPPELLO et al., 2005)

do mesmo programa rodam simultaneamente, cada uma operando no seu bloco de dados. Cada cópia roda um processo BSP independente. No final de cada iteração, os dados para os planos que se encontram nas fronteiras entre os blocos são transmitidos entre os processos adequados e a *superstep* é concluída.

Figura 20 – Modelagem dos blocos



Fonte: Elaborado pelo autor

6.2 Avaliação

A Tabela 6 apresenta os tempos para a execução de 11 processos. É percebido que a demanda para a execução do BSPonP2P é pequena, comparando os cenários i e ii (menor que 4%). A partir disso, os processos são balanceados e, em consequência geram o aumento do valor de α nas etapas de reescalonamento. Sendo assim, isso explica o baixo impacto quando comparados os cenários i e ii. Apesar disso, o BSPonP2P decide que as migrações são inviáveis para qualquer momento, independentemente da quantidade de *Supersteps* executadas.

Tabela 6 – Avaliação de 11 processos, considerando os três cenários (tempo em segundos)

Supersteps	Cenário i	$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
		Cenário ii	Cenário iii	Cenário ii	Cenário iii	Cenário ii	Cenário iii
10	479,17	496,67	502,69	485,97	485,59	480,02	479,17
50	1624,23	1688,86	1685,09	1665,43	1663,61	1664,48	1641,28
100	3055,75	3158,86	3161,55	3147,91	3145,11	3143,98	3099,93
500	14772,9	14941,1	14981,84	14928,7	14855,24	14939	14977,71
1000	29338	29727,4	29680,15	29652,3	29542,58	29676,6	29859,16
2000	58968,4	59191,9	59183,98	59076,9	58975,19	59394,4	59541,62

Fonte: Elaborado pelo autor

Os resultados da execução de 26 processos são apresentados na Tabela 7. Nesse contexto é destacado o ganho obtido na execução de 2000 *supersteps* e $\alpha = 16$, onde foi obtido um ganho de 6% a partir de 14 migrações de processos. O ganho foi obtido mesmo havendo a sobrecarga da execução do serviço de *checkpoint*.

Tabela 7 – Avaliação de 26 processos, considerando os três cenários (tempo em segundos)

Supersteps	Cenário i	$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
		Cenário ii	Cenário iii	Cenário ii	Cenário iii	Cenário ii	Cenário iii
10	335,42	347,67	351,88	340,18	339,91	336,01	335,42
50	1136,96	1182,2	1179,56	1165,8	1164,53	1165,14	1148,9
100	2139,03	2211,2	2083,09	2297,97	2201,58	2357,99	2169,95
500	10341,03	10458,77	10487,29	10450,09	10398,67	10457,3	10484,4
1000	20536,6	20809,18	20776,11	21349,66	20679,81	20773,62	20901,41
2000	43221,2	46761,6	41428,79	43353,83	41282,63	44576,08	40679,13

Fonte: Elaborado pelo autor

Nas Tabelas 8 e 9, são apresentados os dados obtidos na execução de 51 e 89 processos respectivamente. A partir dos dados, é verificada baixa sobrecarga em relação a comparação aos cenários i e iii. A sobrecarga gerada no cenário iii é causado pelo serviço de *checkpoint*. Na Tabela 8, no cenário iii com $\alpha = 16$ o serviço de *checkpoint* demandou 126,12 segundos. Desta forma, caso não houvesse este serviço o desempenho do cenário iii seria melhor do que o cenário ii.

Na Tabela 10, são apresentados os dados referentes às saídas inesperadas da CON. Há a comparação do cenário iii com a execução de 89 processos, 2000 *supersteps* e $\alpha = 16$. Com a saída do nó na *superstep* 9, não houve ganho porque o modelo teve que identificar a falha e

Tabela 8 – Avaliação de 51 processos, considerando os três cenários (tempo em segundos)

Supersteps	Cenário i	$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
		Cenário ii	Cenário iii	Cenário ii	Cenário iii	Cenário ii	Cenário iii
10	268,34	278,14	295,58	272,14	271,93	285,61	268,34
50	909,57	945,76	943,65	932,64	931,62	932,11	919,12
100	1711,22	1768,96	1765,47	1838,38	1761,26	1886,39	1866,16
500	8272,82	8367,02	8389,83	8360,07	8318,93	8365,84	8387,52
1000	18072,21	19647,34	16620,88	18179,72	16543,84	18618,9	18184,23
2000	33022,3	37409,28	33551,32	33083,06	33026,11	33260,86	34143,31

Fonte: Elaborado pelo autor

Tabela 9 – Avaliação de 89 processos, considerando os três cenários (tempo em segundos)

Supersteps	Cenário i	$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
		Cenário ii	Cenário iii	Cenário ii	Cenário iii	Cenário ii	Cenário iii
10	214,67	222,51	248,29	217,71	217,54	242,77	214,67
50	727,66	775,52	754,92	746,11	745,3	745,69	735,29
100	1368,98	1415,17	1522,6	1470,7	1409,01	1509,11	1400,9
500	6618,26	6993,61	6711,86	6638,06	6655,15	6692,67	6710,01
1000	15903,54	16317,88	13296,71	17005,37	13235,08	18295,12	15820,28
2000	26417,84	29927,42	27134,43	26466,45	26420,89	26608,69	27042,11

Fonte: Elaborado pelo autor

iniciar todo o trabalho novamente. Como não houve reescalonamento, não houve *checkpoint*. Já comparando com o ambiente onde houve o erro na *superstep* 1999, o ganho foi de 135%. A última barreira salva havia sido na *superstep* 1016. Como o erro aconteceu na 1999, foi recuperado a partir da 1016.

Tabela 10 – Avaliação da ativação de *checkpoints*

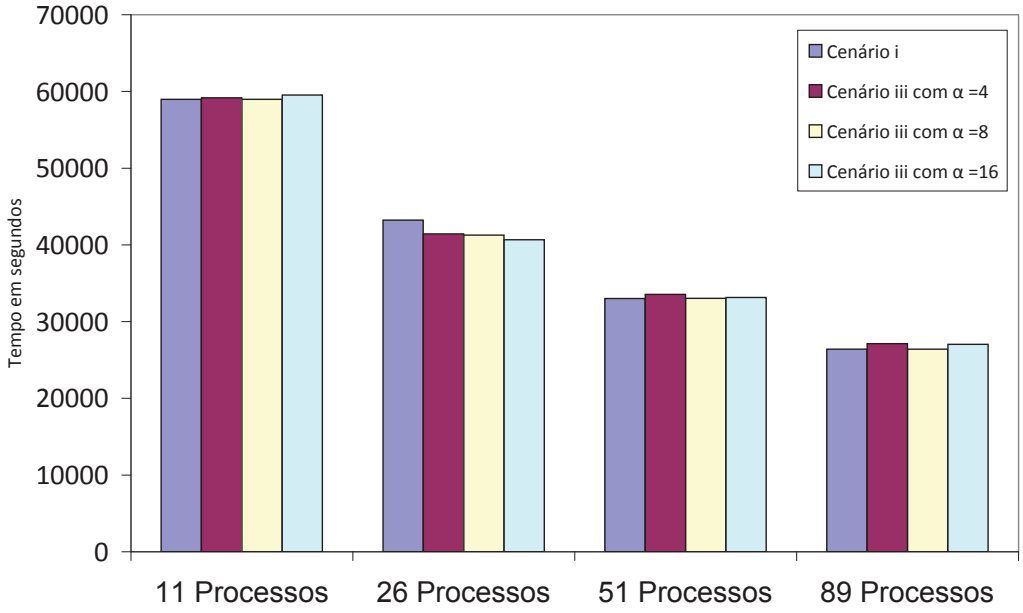
***Supertep* com falha**	**Sem *checkpoint***	**Com *checkpoint***
9	453,92	453,92
49	1684,21	812,66
199	3054,26	1507,37
499	15243,84	6903,89
999	40245,98	15924,92
1999	68742,36	29178,22

Fonte: Elaborado pelo autor

6.3 Análise

O gráfico na Figura 21 demonstra o desempenho do BSPonP2P em relação à execução de 2000 *supersteps*, com migrações e *checkpoint*. Primeiramente, é observado que quanto maior o número de processos, melhores são os resultado alcançados no cenário i. Por outro lado, é visível que esse ganho tende a estabilizar-se em uma situação onde um grande número de processos não implica melhor desempenho. BSPonP2P obteve melhores resultados na execução de 26 processos. Como foi aumentado o número de processos BSP, houve uma diminuição considerável na quantidade de computação atrelada a cada um.

Figura 21 – Análise dos ganhos com migração comparando os cenários i e iii no lançamento de 2000 *supersteps*



Fonte: Elaborado pelo autor

A Figura 21 deixa claro que o BSPonP2P teve pior desempenho em vários casos. Esta perda foi acarretada pela execução do serviço de *checkpoint*. Como pode ser visto na Tabela 10, o *checkpoint* traz um desempenho considerável quando existem inconsistências. Este serviço em ambientes voluntários, como Grades P2P, gera garantia e desempenho para execução. Isto porque o usuário não é obrigado a ficar na CON, sendo assim ele pode sair no meio de uma execução. Caso não houvesse o recurso de *checkpoint*, a aplicação iniciaria os processos do zero, havendo maior demanda de tempo.

Sendo assim, o método Lattice Boltzmann demonstrou que quando há um pequeno número de *supersteps*, não existe tempo para empregar os custos da migração. Essa situação está visível quando houve o teste de 10 e 50 *supersteps*. Por outro lado, quanto maior o cenário, melhor o ganho com a migração dos processos, como pode ser visto no teste com 2000 *supersteps*.

7 CONCLUSÃO

Desafios relacionados à área de sistemas distribuídos estão em constante crescimento, e, consequentemente, estão surgindo diversas abordagens a fim de solucionar esses problemas. Uma delas é criar formas de interligar equipamentos para compartilhar o seu poder computacional e, assim, resolver problemas complexos, porém os equipamentos podem ser dedicados ou não e outra influência forte que impacta é o tipo de usuário: se ele é um usuário comum ou alguém técnico. Para cada tipo de cenário, existe uma topologia diferente. Algumas topologias básicas podem ser citadas: *Clusters*, Grades de Computadores, Grades *Desktop*, P2P e Computação em Nuvem. Porém, dentre as topologias, existem as que foram criadas para aproveitar o que é melhor de cada uma das básicas, como Grades P2P (junção de Grades *Desktop* com P2P) e Computação P2P em Nuvem (P2P com Computação em Nuvem).

A arquitetura Grade P2P foi selecionada para o desenvolvimento desta pesquisa por trabalhar com equipamentos convencionais e não dedicados e porque os usuários não precisam de um preparo específico para utilizar a rede. A seleção desta arquitetura foi devido à percepção do desperdício de computação que, diariamente, há nos lares ou empresas, pois equipamentos são comprados e, na maioria das vezes, são pouco utilizados, visto que, em grande parte do dia, apenas executam navegadores, ferramentas de digitação de texto, planilhas e pequenos aplicativos. A maioria das aplicações consome pouco do equipamento, acarretando desperdício. Além disso, os equipamentos não são corretamente dimensionados para a compra porque eles são adquiridos com grande capacidade para execução de pequenas tarefas. Por isso, o aprimoramento é dado por meio da divisão em pequenas partes de uma necessidade (processo) e da distribuição delas entre os participantes, mas, visando ao controle desses subprocessos, o modelo BSP será aplicado por causa de suas características, como barreiras nas *supersteps*, sincronismo e comunicação.

Algo importante na topologia utilizada é sua grande dinamicidade e heterogeneidade, e, como reflexo dessas características, a utilização de migração é de suma importância. Como BSP trabalha com várias *supersteps*, o modelo utiliza uma variável de ambiente denominada α para saber em qual barreira da *superstep* deve haver avaliações. Ela foi incorporada para que não houvesse sobrecarga no sistema devido às comunicações. As avaliações utilizarão o ínidice PM para determinar migrações, afim de aprimorar as execuções. O índice utiliza métricas como memória, computação, comunicação e dados de equipamentos, sendo que o maior valor de PM indica o melhor grupo para execução.

Sendo assim, o emprego de métricas que fazem uso de dados referentes ao ambiente, às aplicações BSP e às redes Grades P2P permite a utilização de recursos computacionais presentes em lares para a execução de computação útil. A combinação dessas técnicas, além de permitir o compartilhamento, traz eficiência para o sistema.

7.1 Contribuições

Retomando a questão *Problema* indicada na Seção 1.1: *Como explorar computação colaborativa em Grades P2P para execução de aplicações BSP com eficiência?*. A partir do modelo BSPonP2P é atendida a questão porque ele cria uma rede que utiliza uma forma híbrida (rede estruturada e não estruturada) a fim de não sobrecarregar o sistema com troca de mensagens. Esse tipo de arquitetura permite também o controle sobre entradas e saídas do ambiente de maneira fácil. O resultado da eficiência é dado, primeiramente avaliando a comparação entre a simples execução da aplicação (sem o BSPonP2P) com a execução do modelo BSPonP2P com migrações, pois houve um aumento menor que 4% no tempo total de execução. Além disso, na execução de 26 processos com 2000 *supersteps* e $\alpha = 16$, obteve-se um ganho de 6% a partir de 24 migrações de processos.

Na Tabela 11, ficam evidentes algumas contribuições que o modelo BSPonP2P traz:

Tabela 11 – Avaliação entre estruturas e algoritmos de balanceamento

Trabalho	**Arquitetura**	**Modelo de Programação Paralela**	**Serviços**	**Método**	**Observação**
Sentís et al. (2014)	Grades P2P	Mestre/Escravo	Migração	Métricas Computacionais	Computação e Memória
Khayyat et al. (2013)	Grades de *Desktop*	BSP	Migração	Métricas Computacionais	Computação
Shudo, Tanaka e Sekiguchi (2005)	Grades P2P	Mestre/Escravo	Migração	Métricas Computacionais	Computação
Camargo, Castor e Kon (2010)	Grades P2P	BSP	Migração e Replicação de dados	Métricas Computacionais	Computação
Son, Lee e Youn (2010)	Grades de *Desktop*	Mestre/Escravo	Migração	Métricas Computacionais	Computação
Balasubramaniam et al. (2012)	Grades de *Desktop*	*Bag of Tasks*	Não Informado	Métricas Computacionais	Computação
Godfrey et al. (2004)	Grades P2P	Mestre/Escravo	Migração e Replicação de Dados	Métricas Computacionais	Computação
Wu, Tian e Ng (2006)	Grades P2P	Mestre/Escravo	Migração e Replicação de Dados	Métricas Computacionais	Computação
Leite et al. (2012)	Grades P2P	*Bag of Tasks*	Migração e Replicação de Computação	Política de Roubos de Trabalhos	Computação
Anceaume et al. (2006)	Grades P2P	*Bag of Tasks*	Não Informado	Métricas Computacionais	Computação
Rosa Righi et al. (2011)	Grade de Computadores	BSP	Migração	Métricas Computacionais	Computação, Comunicação e Memória
BSPonP2P	Grades P2P	BSP	Migração	Métricas Computacionais	Computação, Comunicação, Memória e Dados de Ambiente

Fonte: Elaborado pelo autor

- Utilização de métricas como Computação, Comunicação, Memória e Dados de Ambiente. Poucos autores trabalham em uma avaliação complexa do ambiente. Conforme pode ser

visto, é trabalhado, na grande maioria das vezes, apenas com Computação. A avaliação utilizando métricas como Comunicação, Memória e Dados de Ambiente permite uma verificação mais precisa do ambiente;

- A utilização de Grades P2P combinadas com métricas complexas. Mesmo Grades P2P serem comuns, existem poucos trabalhos que utilizam este tipo de rede combinado com avaliação mais complexa do ambiente. Além disso, outro fator que destaca o modelo BSPonP2P é a utilização de duas topologias de rede (estruturada e não estruturada);

- Utilização de *checkpoints*. Este serviço agrega segurança ao modelo, uma preocupação comum em ambientes dinâmicos. Porém, utilizá-lo combinado com aplicações BSP e avaliações periódicas não é algo muito presente.

7.2 Trabalhos Futuros

É importante salientar que esta dissertação abre caminho para uma série de trabalhos futuros. Dentre estas possibilidades, destacam-se:

- Identificação de uma função de distribuição de probabilidade para aprimorar o índice que avalia os dados de ambiente. A utilização de processos estocásticos, trará melhores benefícios do que o uso de uma simples média aritmética;

- Criação de replicação de computação. Como somente são criados *checkpoints* em barreiras α e há a possibilidade de saídas inesperadas, é possível uma demanda alta para a recuperação de dados. Para minimizar essa perda, a utilização de replicação de computação é algo que auxiliaria na solução do problema;

- Construção da CON. A CON é feita a partir da entrada dos nós na rede. Porém, a CON poderia ter melhor desempenho, caso houvesse uma reavaliação do ambiente de forma que ela fosse reestruturada a fim de diminuir a latência.

REFERÊNCIAS

ALTUHAFI, A.; RAMADASS, S.; CHONG, Y.-W. Concepts and types of peer-to-peer network topology for live video streaming. In: RFID-TECHNOLOGIES AND APPLICATIONS (RFID-TA), 2013 IEEE INTERNATIONAL CONFERENCE ON, 2013. **Anais...** [S.l.: s.n.], 2013. p. 1–4.

ANCEAUME, E.; GRADINARIU, M.; DATTA, A.; SIMON, G.; VIRGILLITO, A. A Semantic Overlay for Self- Peer-to-Peer Publish/Subscribe. In: DISTRIBUTED COMPUTING SYSTEMS, 2006. ICDCS 2006. 26TH IEEE INTERNATIONAL CONFERENCE ON, 2006. **Anais...** [S.l.: s.n.], 2006. p. 22–22.

ANDERSON, D. P. BOINC: a system for public-resource computing and storage. In: IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING, 5., 2004, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2004. p. 4–10. (GRID '04).

ANDERSON, D. P.; COBB, J.; KORPELA, E.; LEBOFSKY, M.; WERTHIMER, D. SETI@home: an experiment in public-resource computing. **Commun. ACM**, New York, NY, USA, v. 45, n. 11, p. 56–61, Nov. 2002.

ANDROUTSELLIS-THEOTOKIS, S.; SPINELLIS, D. A survey of peer-to-peer content distribution technologies. **ACM Comput. Surv.**, New York, NY, USA, v. 36, n. 4, p. 335–371, Dec. 2004.

BALASUBRAMANIAM, M.; SUKHIJA, N.; CIORBA, F.; BANICESCU, I.; SRIVASTAVA, S. Towards the Scalability of Dynamic Loop Scheduling Techniques via Discrete Event Simulation. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2012 IEEE 26TH INTERNATIONAL, 2012. **Anais...** [S.l.: s.n.], 2012. p. 1343–1351.

BANDARA, H.; JAYASUMANA, A. Collaborative applications over peer-to-peer systems–challenges and solutions. **Peer-to-Peer Networking and Applications**, [S.l.], v. 6, n. 3, p. 257–276, 2013.

BASU, A.; FLEMING, S.; STANIER, J.; NAICKEN, S.; WAKEMAN, I.; GURBANI, V. K. The state of peer-to-peer network simulators. **ACM Comput. Surv.**, New York, NY, USA, v. 45, n. 4, p. 46:1–46:25, Aug. 2013.

BOBELIN, L.; LEGRAND, A.; MáRQUEZ, D. A. G.; NAVARRO, P.; QUINSON, M.; SUTER, F.; THIERY, C. Scalable Multi-purpose Network Representation for Large Scale Distributed System Simulation. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER, CLOUD AND GRID COMPUTING (CCGRID 2012), 2012., 2012, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2012. p. 220–227. (CCGRID '12).

BRETZKE, H.; VASSILEVA, J. Motivating Cooperation on Peer to Peer Networks. In: BRUSILOVSKY, P.; CORBETT, A.; ROSIS, F. de (Ed.). **User Modeling 2003**. [S.l.]: Springer Berlin Heidelberg, 2003. p. 218–227. (Lecture Notes in Computer Science, v. 2702).

BYERS, J.; CONSIDINE, J.; MITZENMACHER, M. Simple Load Balancing for Distributed Hash Tables. In: KAASHOEK, M.; STOICA, I. (Ed.). **Peer-to-Peer Systems II**. [S.l.]: Springer Berlin Heidelberg, 2003. p. 80–87. (Lecture Notes in Computer Science, v. 2735).

CAMARGO, R.; CASTOR, F.; KON, F. Reliable management of checkpointing and application data in opportunistic grids. **Journal of the Brazilian Computer Society**, [S.l.], v. 16, n. 3, p. 177–190, 2010.

CAPPELLO, F.; CARON, E.; DAYDE, M.; DESPREZ, F.; JEGOU, Y.; PRIMET, P.; JEANNOT, E.; LANTERI, S.; LEDUC, J.; MELAB, N.; MORNET, G.; NAMYST, R.; QUETIER, B.; RICHARD, O. Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In: GRID COMPUTING, 2005. THE 6TH IEEE/ACM INTERNATIONAL WORKSHOP ON, 2005. **Anais...** [S.l.: s.n.], 2005. p. 8 pp.–.

CASANOVA, H.; LEGRAND, A.; QUINSON, M. SimGrid: a generic framework for large-scale distributed experiments. In: TENTH INTERNATIONAL CONFERENCE ON COMPUTER MODELING AND SIMULATION, 2008, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2008. p. 126–131. (UKSIM '08).

CASAVANT, T.; KUHL, J. A taxonomy of scheduling in general-purpose distributed computing systems. **Software Engineering, IEEE Transactions on**, [S.l.], v. 14, n. 2, p. 141–154, 1988.

CASTELLà, D.; BARRI, I.; RIUS, J.; GINé, F.; SOLSONA, F.; GUIRADO, F. CoDiP2P: a peer-to-peer architecture for sharing computing resources. In: CORCHADO, J.; RODRíGUEZ, S.; LLINAS, J.; MOLINA, J. (Ed.). **International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)**. [S.l.]: Springer Berlin Heidelberg, 2009. p. 293–303. (Advances in Soft Computing, v. 50).

CASTELLà, D.; BLANCO, H.; GINé, F.; SOLSONA, F. Combining Hilbert SFC and Bruijn Graphs for Searching Computing Markets in a P2P System. In: D'AMBRA, P.; GUARRACINO, M.; TALIA, D. (Ed.). **Euro-Par 2010 - Parallel Processing**. [S.l.]: Springer Berlin Heidelberg, 2010. p. 471–483. (Lecture Notes in Computer Science, v. 6271).

CASTRO, H.; VILLAMIZAR, M.; SOTELO, G.; DIAZ, C.; PECERO, J.; BOUVRY, P. Green flexible opportunistic computing with task consolidation and virtualization. **Cluster Computing**, [S.l.], v. 16, n. 3, p. 545–557, 2013.

CELAYA, J.; MARCHAL, L. A Fair Decentralized Scheduler for Bag-of-Tasks Applications on Desktop Grids. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2010 10TH IEEE/ACM INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p. 538–541.

CERIN, C.; TAKOUDJOU, A. BOINC as a Service for the SlapOS Cloud: tools and methods. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS PHD FORUM (IPDPSW), 2013 IEEE 27TH INTERNATIONAL, 2013. **Anais...** [S.l.: s.n.], 2013. p. 974–983.

CHOI, S.; KIM, H.; BYUN, E.; HWANG, C. **A Taxonomy of Desktop Grid Systems Focusing on Scheduling**. 2006.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Sistemas Distribuídos**: conceitos e projeto. [S.l.]: Bookman, 2007.

EAGER, D. L.; LAZOWSKA, E. D.; ZAHORJAN, J. A comparison of receiver-initiated and sender-initiated adaptive load sharing (extended abstract). In: ACM SIGMETRICS

CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 1985., 1985, New York, NY, USA. **Proceedings...** ACM, 1985. p. 1–3. (SIGMETRICS '85).

ERNEMANN, C.; HAMSCHER, V.; SCHWIEGELSHOHN, U.; YAHYAPOUR, R.; STREIT, A. On Advantages of Grid Computing for Parallel Job Scheduling. In: CLUSTER COMPUTING AND THE GRID, 2002. 2ND IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2002. **Anais...** [S.l.: s.n.], 2002. p. 39–39.

ERNST-DESMULIER, J.-B.; BOURGEOIS, J.; NGO, M. T.; SPIES, F.; VERBEKE, J. Simulating and optimizing a peer-to-peer computing framework. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2006. IPDPS 2006. 20TH INTERNATIONAL, 2006. **Anais...** [S.l.: s.n.], 2006. p. 8 pp.–.

GODFREY, B.; LAKSHMINARAYANAN, K.; SURANA, S.; KARP, R.; STOICA, I. Load balancing in dynamic structured P2P systems. In: INFOCOM 2004. TWENTY-THIRD ANNUALJOINT CONFERENCE OF THE IEEE COMPUTER AND COMMUNICATIONS SOCIETIES, 2004. **Anais...** [S.l.: s.n.], 2004. v. 4, p. 2253–2262 vol.4.

GOOD, N. S.; KREKELBERG, A. Usability and Privacy: a study of kazaa p2p file-sharing. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2003, New York, NY, USA. **Proceedings...** ACM, 2003. p. 137–144. (CHI '03).

GU, Y.; TANG, K.-M.; CHEN, L. Design and implementation of an event-based P2P simulator: e-simulator. In: MACHINE LEARNING AND CYBERNETICS, 2009 INTERNATIONAL CONFERENCE ON, 2009. **Anais...** [S.l.: s.n.], 2009. v. 3, p. 1359–1363.

HALIM, R.; ABIDIN, S.; LATIP, R. Grid task scheduling in P2P Desktop Grids. In: BUSINESS ENGINEERING AND INDUSTRIAL APPLICATIONS COLLOQUIUM (BEIAC), 2012 IEEE, 2012. **Anais...** [S.l.: s.n.], 2012. p. 150–155.

HILL, J. M. D.; MCCOLL, B.; STEFANESCU, D. C.; GOUDREAU, M. W.; LANG, K.; RAO, S. B.; SUEL, T.; TSANTILAS, T.; BISSELING, R. H. BSPlib: the bsp programming library. **Parallel Comput.**, Amsterdam, The Netherlands, The Netherlands, v. 24, n. 14, p. 1947–1980, Dec. 1998.

HUANG, Y.; QIAN, D.; LUAN, Z.; WANG, Y.; WU, Z.; YAN, B. EOMT: a master-slave task scheduling strategy for grid environment. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS, 2008. HPCC '08. 10TH IEEE INTERNATIONAL CONFERENCE ON, 2008. **Anais...** [S.l.: s.n.], 2008. p. 226–233.

IKEUCHI, M.; IGARASHI, H.; OKADA, K.; TSUKAYA, H. Acropetal leaflet initiation of Eschscholzia californica is achieved by constant spacing of leaflets and differential growth of leaf. **Planta**, [S.l.], v. 240, n. 1, p. 125–135, 2014.

IORGA, R.; BORCOCI, E.; MIRUTA, R.; PINTO, A.; CARNEIRO, G.; CALCADA, T. Management driven hybrid multicast framework for content aware networks. **Communications Magazine, IEEE**, [S.l.], v. 52, n. 1, p. 158–165, January 2014.

JACKSON, G.; KELEHER, P.; SUSSMAN, A. Decentralized Scheduling and Load Balancing for Parallel Programs. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2014 14TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2014. **Anais...** [S.l.: s.n.], 2014. p. 324–333.

JAIN, S.; CHAUDHARY, J. New fault tolerant scheduling algorithm implemented using check pointing in grid computing environment. In: OPTIMIZATION, RELIABILTY, AND INFORMATION TECHNOLOGY (ICROIT), 2014 INTERNATIONAL CONFERENCE ON, 2014. **Anais...** [S.l.: s.n.], 2014. p. 393–396.

JAISWAL, D.; MISTRY, S.; MUKHERJEE, A.; MUKHERJEE, N. Efficient Dynamic Service Provisioning over Distributed Resources Using Chord. In: SIGNAL-IMAGE TECHNOLOGY INTERNET-BASED SYSTEMS (SITIS), 2013 INTERNATIONAL CONFERENCE ON, 2013. **Anais...** [S.l.: s.n.], 2013. p. 257–264.

KHAYYAT, Z.; AWARA, K.; ALONAZI, A.; JAMJOOM, H.; WILLIAMS, D.; KALNIS, P. Mizan: a system for dynamic load balancing in large-scale graph processing. In: ACM EUROPEAN CONFERENCE ON COMPUTER SYSTEMS, 8., 2013, New York, NY, USA. **Proceedings...** ACM, 2013. p. 169–182. (EuroSys '13).

KIJSIPONGSE, E.; U-RUEKOLAN, S. Scaling HPC Clusters with volunteer computing for data intensive applications. In: COMPUTER SCIENCE AND SOFTWARE ENGINEERING (JCSSE), 2013 10TH INTERNATIONAL JOINT CONFERENCE ON, 2013. **Anais...** [S.l.: s.n.], 2013. p. 138–142.

KOLICI, V.; MATSUO, K.; BAROLLI, L.; XHAFA, F.; DURRESI, A.; MIHO, R. Application of a JXTA-overlay P2P system for end-device control and e-learning. **Multimedia Tools and Applications**, [S.l.], v. 53, n. 2, p. 371–389, 2011.

KONDO, D.; TAUFER, M.; BROOKS, C. L.; III; CASANOVA, H.; CHIEN, A. A. Characterizing and Evaluating Desktop Grids: an empirical study. In: IN PROCEEDINGS OF THE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS'04, 2004. **Anais...** [S.l.: s.n.], 2004.

KUPER, L.; TURON, A.; KRISHNASWAMI, N. R.; NEWTON, R. R. Freeze After Writing: quasi-deterministic parallel programming with lvars. In: ACM SIGPLAN-SIGACT SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, 41., 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 257–270. (POPL '14).

LAREDO, J.; BOUVRY, P.; GONZáLEZ, D.; VEGA, F. Fernández de; ARENAS, M.; MERELO, J.; FERNANDES, C. Designing robust volunteer-based evolutionary algorithms. **Genetic Programming and Evolvable Machines**, [S.l.], v. 15, n. 3, p. 221–244, 2014.

LEITE, A.; MENDES, H.; WEIGANG, L.; MELO, A.; BOUKERCHE, A. An architecture for P2P bag-of-tasks execution with multiple task allocation policies in desktop grids. **Cluster Computing**, [S.l.], v. 15, n. 4, p. 351–361, 2012.

LI, R.-H.; YU, J. X.; HUANG, X.; CHENG, H. Random-walk domination in large graphs. In: DATA ENGINEERING (ICDE), 2014 IEEE 30TH INTERNATIONAL CONFERENCE ON, 2014. **Anais...** [S.l.: s.n.], 2014. p. 736–747.

LIN, L.; KOYANAGI, K.; TSUCHIYA, T.; MIYOSAWA, T.; HIROSE, H. Improving routing load balance on Chord. In: ADVANCED COMMUNICATION TECHNOLOGY (ICACT), 2014 16TH INTERNATIONAL CONFERENCE ON, 2014. **Anais...** [S.l.: s.n.], 2014. p. 733–738.

LIN, S.-J.; HUANG, M.-C.; LAI, K.-C.; HUANG, K.-C. Design and Implementation of Job Migration Policies in P2P Grid Systems. In: ASIA-PACIFIC SERVICES COMPUTING CONFERENCE, 2008. APSCC '08. IEEE, 2008. **Anais...** [S.l.: s.n.], 2008. p. 75–80.

LIU, Y.; LI, Y.; YANG, L.; XIONG, N.; ZHU, L.; XU, K. The resource locating strategy based on sub-domain hybrid P2P network model. In: PARALLEL DISTRIBUTED PROCESSING, WORKSHOPS AND PHD FORUM (IPDPSW), 2010 IEEE INTERNATIONAL SYMPOSIUM ON, 2010. **Anais...** [S.l.: s.n.], 2010. p. 1–8.

LOULERGUE, F. Optimizing bulk synchronous parallel ML. In: SOFTWARE ENGINEERING, ARTIFICIAL INTELLIGENCE, NETWORKING AND PARALLEL/DISTRIBUTED COMPUTING, 2005 AND FIRST ACIS INTERNATIONAL WORKSHOP ON SELF-ASSEMBLING WIRELESS NETWORKS. SNPD/SAWN 2005. SIXTH INTERNATIONAL CONFERENCE ON, 2005. **Anais...** [S.l.: s.n.], 2005. p. 294–299.

LYKOURGIOTIS, A.; BIRKOS, K.; DAGIUKLAS, T.; EKMEKCIOGLU, E.; DOGAN, S.; YILDIZ, Y.; POLITIS, I.; TANIK, G.; DEMIRTAS, B.; KONDOZ, A.; KOTSOPOULOS, S. Hybrid broadcast and broadband networks convergence for immersive TV applications. **Wireless Communications, IEEE**, [S.l.], v. 21, n. 3, p. 62–69, June 2014.

MA, L.; HE, J.; MA, D. A Localization Algorithm Based on Virtual Beacon Nodes for Wireless Sensor Network. In: SUN, L.; MA, H.; HONG, F. (Ed.). **Advances in Wireless Sensor Networks**. [S.l.]: Springer Berlin Heidelberg, 2014. p. 271–280. (Communications in Computer and Information Science, v. 418).

MALEWICZ, G.; AUSTERN, M. H.; BIK, A. J.; DEHNERT, J. C.; HORN, I.; LEISER, N.; CZAJKOWSKI, G. Pregel: a system for large-scale graph processing. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 2010., 2010, New York, NY, USA. **Proceedings...** ACM, 2010. p. 135–146. (SIGMOD '10).

MATSUO, K.; BAROLLI, L.; XHAFA, F.; KOLICI, V.; KOYAMA, A.; DURRESI, A.; MIHO, R. Implementation of an E-learning System Using P2P, Web and Sensor Technologies. In: ADVANCED INFORMATION NETWORKING AND APPLICATIONS, 2009. AINA '09. INTERNATIONAL CONFERENCE ON, 2009. **Anais...** [S.l.: s.n.], 2009. p. 800–807.

MAYMOUNKOV, P.; MAZIèRES, D. Kademlia: a peer-to-peer information system based on the xor metric. In: DRUSCHEL, P.; KAASHOEK, F.; ROWSTRON, A. (Ed.). **Peer-to-Peer Systems**. [S.l.]: Springer Berlin Heidelberg, 2002. p. 53–65. (Lecture Notes in Computer Science, v. 2429).

MELAB, N.; MEZMAZ, M.-S.; TALBI, E.-G. Parallel Hybrid Multi-Objective Island Model in Peer-to-Peer Environment. In: PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2005. PROCEEDINGS. 19TH IEEE INTERNATIONAL, 2005. **Anais...** [S.l.: s.n.], 2005. p. 190b–190b.

MIRIAM, D. D. H.; EASWARAKUMAR, K. S. HPGRID: a novel architectural model for resource management systems. In: INTERNATIONAL CONFERENCE ON COMMUNICATION, COMPUTING & SECURITY, 2011., 2011, New York, NY, USA. **Proceedings...** ACM, 2011. p. 427–432. (ICCCS '11).

MOHAMED, N.; AL-JAROODI, J.; JIANG, H. DDOps: dual-direction operations for load balancing on non-dedicated heterogeneous distributed systems. **Cluster Computing**, [S.l.], v. 17, n. 2, p. 503–528, 2014.

MONTRESOR, A.; JELASITY, M. PeerSim: a scalable p2p simulator. In: PEER-TO-PEER COMPUTING, 2009. P2P '09. IEEE NINTH INTERNATIONAL CONFERENCE ON, 2009. **Anais...** [S.l.: s.n.], 2009. p. 99–100.

NAOR, M.; WIEDER, U. Novel Architectures for P2P Applications: the continuous-discrete approach. In: FIFTEENTH ANNUAL ACM SYMPOSIUM ON PARALLEL ALGORITHMS AND ARCHITECTURES, 2003, New York, NY, USA. **Proceedings...** ACM, 2003. p. 50–59. (SPAA '03).

NATARAJAN, C. Improving Service Performance in Cloud Computing with Network Memory Virtualization. In: BOONKRONG, S.; UNGER, H.; MEESAD, P. (Ed.). **Recent Advances in Information and Communication Technology**. [S.l.]: Springer International Publishing, 2014. p. 233–243. (Advances in Intelligent Systems and Computing, v. 265).

NESMACHNOW, S.; DORRONSORO, B.; PECERO, J.; BOUVRY, P. Energy-Aware Scheduling on Multicore Heterogeneous Grid Computing Systems. **Journal of Grid Computing**, [S.l.], v. 11, n. 4, p. 653–680, 2013.

OGATA, Y.; SPAHO, E.; MATSUO, K.; BAROLLI, L.; ARNEDO-MORENO, J.; XHAFA, F. JXTA-Overlay P2P Platform and Its Application for Robot Control. In: NETWORK-BASED INFORMATION SYSTEMS (NBIS), 2010 13TH INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p. 133–138.

OGATA, Y.; SPAHO, E.; MATSUO, K.; BAROLLI, L.; ARNEDO-MORENO, J.; XHAFA, F. JXTA-Overlay P2P Platform and Its Application for Robot Control. **2012 15th International Conference on Network-Based Information Systems**, Los Alamitos, CA, USA, v. 0, p. 133–138, 2010.

PANDE, V. S.; BAKER, I.; CHAPMAN, J.; ELMER, S. P.; KHALIQ, S.; LARSON, S. M.; RHEE, Y. M.; SHIRTS, M. R.; SNOW, C. D.; SORIN, E. J.; ZAGROVIC, B. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. **Biopolymers**, Department of Chemistry, Stanford University, Stanford, CA 94305-5080; Biophysics Program, Stanford University, Stanford, CA 94305-5080; Department of Structural Biology, Stanford University, Stanford, CA 94305-5080; Stanford Synchrotron Radiation Laboratory, Stanford University, Stanford, CA 94305-5080; Department of Computer Science, Stanford University, Stanford, CA 94305-5080, v. 68, n. 1, p. 91–109, 2003.

PING, T.; SODHY, G.; YONG, C.; HARON, F.; BUYYA, R. A Market-Based Scheduler for JXTA-Based Peer-to-Peer Computing System. In: LAGAN, A.; GAVRILOVA, M.; KUMAR, V.; MUN, Y.; TAN, C.; GERVASI, O. (Ed.). **Computational Science and Its Applications ? ICCSA 2004**. [S.l.]: Springer Berlin Heidelberg, 2004. p. 147–157. (Lecture Notes in Computer Science, v. 3046).

QURESHI, M.; DEHNAVI, M.; MIN-ALLAH, N.; QURESHI, M.; HUSSAIN, H.; RENTIFIS, I.; TZIRITAS, N.; LOUKOPOULOS, T.; KHAN, S.; XU, C.-Z.; ZOMAYA, A. Survey on Grid Resource Allocation Mechanisms. **Journal of Grid Computing**, [S.l.], p. 1–43, 2014.

RAJ, J.; HARIKUMAR, R. A dynamic overlay approach for mobility maintenance in personal communication networks. **Peer-to-Peer Networking and Applications**, [S.l.], v. 7, n. 2, p. 118–128, 2014.

RATNASAMY, S.; FRANCIS, P.; HANDLEY, M.; KARP, R.; SHENKER, S. A Scalable Content-Addressable Network. In: IN PROC. ACM SIGCOMM 2001, 2001. **Anais...** [S.l.: s.n.], 2001. p. 161–172.

RIGHI, R. d. R. **Process Migration in Grid Computing**. x: LAP LAMPERT Academic Publishing, 2012.

RIPEANU, M. Peer-to-peer architecture case study: gnutella network. In: PEER-TO-PEER COMPUTING, 2001. PROCEEDINGS. FIRST INTERNATIONAL CONFERENCE ON, 2001. **Anais...** [S.l.: s.n.], 2001. p. 99–100.

ROSA RIGHI, R. da; GRAEBIN, L.; AVILA, R.; NAVAUX, P.; PILLA, L. Combining Multiple Metrics to Control BSP Process Rescheduling in Response to Resource and Application Dynamics. In: PARALLEL AND DISTRIBUTED SYSTEMS (ICPADS), 2011 IEEE 17TH INTERNATIONAL CONFERENCE ON, 2011. **Anais...** [S.l.: s.n.], 2011. p. 72–79.

ROWSTRON, A.; DRUSCHEL, P. Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: GUERRAOUI, R. (Ed.). **Middleware 2001**. [S.l.]: Springer Berlin Heidelberg, 2001. p. 329–350. (Lecture Notes in Computer Science, v. 2218).

SCHEPKE, C.; MAILLARD, N. Performance Improvement of the Parallel Lattice Boltzmann Method Through Blocked Data Distributions. In: COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 2007. SBAC-PAD 2007. 19TH INTERNATIONAL SYMPOSIUM ON, 2007. **Anais...** [S.l.: s.n.], 2007. p. 71–78.

SENTíS, J.; SOLSONA, F.; CASTELLà, D.; RIUS, J. DisCoP2P: an efficient p2p computing overlay. **The Journal of Supercomputing**, [S.l.], v. 68, n. 2, p. 557–573, 2014.

SHAH, B.; KIM, K.-I. Towards Enhanced Searching Architecture for Unstructured Peer-to-Peer Over Mobile Ad Hoc Networks. **Wireless Personal Communications**, [S.l.], v. 77, n. 2, p. 1167–1189, 2014.

SHAH, R.; VEERAVALLI, B.; MISRA, M. On the Design of Adaptive and Decentralized Load Balancing Algorithms with Load Estimation for Computational Grid Environments. **Parallel and Distributed Systems, IEEE Transactions on**, [S.l.], v. 18, n. 12, p. 1675–1686, 2007.

SHUDO, K.; TANAKA, Y.; SEKIGUCHI, S. P3: p2p-based middleware enabling transfer and aggregation of computational resources. In: CLUSTER COMPUTING AND THE GRID, 2005. CCGRID 2005. IEEE INTERNATIONAL SYMPOSIUM ON, 2005. **Anais...** [S.l.: s.n.], 2005. v. 1, p. 259–266 Vol. 1.

SON, B. H.; LEE, S. woo; YOUN, H.-Y. Prediction-Based Dynamic Load Balancing Using Agent Migration for Multi-agent System. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS (HPCC), 2010 12TH IEEE INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. p. 485–490.

STAINFORTH, D.; KETTLEBOROUGH, J.; MARTIN, A.; SIMPSON, A.; GILLIS, R.; AKKAS, A.; GAULT, R.; COLLINS, M.; GAVAGHAN, D.; ALLEN, M. Climateprediction.net: design principles for public-resource modeling research. In: IN 14TH IASTED INTERNATIONAL CONFERENCE PARALLEL AND DISTRIBUTED COMPUTING AND SYSTEMS, 2002. **Anais...** [S.l.: s.n.], 2002. p. 32–38.

STOICA, I.; MORRIS, R.; LIBEN-NOWELL, D.; KARGER, D.; KAASHOEK, M.; DABEK, F.; BALAKRISHNAN, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. **Networking, IEEE/ACM Transactions on**, [S.l.], v. 11, n. 1, p. 17–32, 2003.

TOMAS, L.; CAMINERO, B.; CARRION, C. Opportunistic energy-aware rescheduling in desktop grid environments. In: HIGH PERFORMANCE COMPUTING AND SIMULATION (HPCS), 2013 INTERNATIONAL CONFERENCE ON, 2013. **Anais...** [S.l.: s.n.], 2013. p. 178–185.

TRAVERSO, S.; ABENI, L.; BIRKE, R.; KIRALY, C.; LEONARDI, E.; LO CIGNO, R.; MELLIA, M. Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: design and experimental comparison. In: 2014. **Anais...** [S.l.: s.n.], 2014. v. PP, n. 99, p. 1–1.

TRUNFIO, P.; TALIA, D.; FRAGOPOULOU, P.; PAPADAKIS, C.; MORDACCHINI, M.; PENNANEN, M.; POPOV, K.; VLASSOV, V.; HARIDI, S. Peer-to-Peer Models for Resource Discovery on Grids. In: COREGRID WORKSHOP ON GRID AND PEER TO PEER SYSTEMS ARCHITECTURE, 2., 2006, Paris, France. **Proceedings...** [S.l.: s.n.], 2006.

UMEZAKI, K.; SPAHO, E.; BAROLLI, L.; XHAFA, F.; BAROLLI, V.; IWASHIGE, J. A Fuzzy-Based System to Evaluate the Peer Reliability in JXTA-Overlay P2P. In: COMPLEX, INTELLIGENT, AND SOFTWARE INTENSIVE SYSTEMS (CISIS), 2013 SEVENTH INTERNATIONAL CONFERENCE ON, 2013. **Anais...** [S.l.: s.n.], 2013. p. 111–116.

VALIANT, L. G. A bridging model for parallel computation. **Commun. ACM**, New York, NY, USA, v. 33, n. 8, p. 103–111, Aug. 1990.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A Break in the Clouds: towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v. 39, n. 1, p. 50–55, Dec. 2008.

VYAS, R.; MAHETA, H.; DABHI, V.; PRAJAPATI, H. Load balancing using process migration for linux based distributed system. In: ISSUES AND CHALLENGES IN INTELLIGENT COMPUTING TECHNIQUES (ICICT), 2014 INTERNATIONAL CONFERENCE ON, 2014. **Anais...** [S.l.: s.n.], 2014. p. 248–252.

WANG, X.; YIN, Y.; YU, H. Finding Collisions in the Full SHA-1. In: SHOUP, V. (Ed.). **Advances in Cryptology – CRYPTO 2005**. [S.l.]: Springer Berlin Heidelberg, 2005. p. 17–36. (Lecture Notes in Computer Science, v. 3621).

WOUNGANG, I.; TSENG, F.-H.; LIN, Y.-H.; CHOU, L.-D.; CHAO, H.-C.; OBAIDAT, M. MR-Chord: improved chord lookup performance in structured mobile p2p networks. In: 2014. **Anais...** [S.l.: s.n.], 2014. v. PP, n. 99, p. 1–9.

WU, D.; TIAN, Y.; NG, K.-W. On the Effectiveness of Migration-based Load Balancing Strategies in DHT Systems. In: COMPUTER COMMUNICATIONS AND NETWORKS, 2006. ICCCN 2006. PROCEEDINGS.15TH INTERNATIONAL CONFERENCE ON, 2006. **Anais...** [S.l.: s.n.], 2006. p. 405–410.

YILDIZ, O.; DORIER, M.; IBRAHIM, S.; ANTONIU, G. A Performance and Energy Analysis of I/O Management Approaches for Exascale Systems. In: SIXTH INTERNATIONAL WORKSHOP ON DATA INTENSIVE DISTRIBUTED COMPUTING, 2014, New York, NY, USA. **Proceedings...** ACM, 2014. p. 35–40. (DIDC '14).

YING, G.; JIANG, Z. A load-balancing structured store model on P2P network. In: COMPUTER ENGINEERING AND TECHNOLOGY (ICCET), 2010 2ND INTERNATIONAL CONFERENCE ON, 2010. **Anais...** [S.l.: s.n.], 2010. v. 3, p. V3–236–V3–239.

YOON, D.; CHO, K.; LEE, J. Bit error probability of M-ary quadrature amplitude modulation. In: VEHICULAR TECHNOLOGY CONFERENCE, 2000. IEEE-VTS FALL VTC 2000. 52ND, 2000. **Anais...** [S.l.: s.n.], 2000. v. 5, p. 2422–2427 vol.5.

YU, J.; BUYYA, R. A taxonomy of scientific workflow systems for grid computing. **SIGMOD Rec.**, New York, NY, USA, v. 34, n. 3, p. 44–49, Sept. 2005.

YZELMAN, A.; BISSELING, R.; ROOSE, D.; MEERBERGEN, K. MulticoreBSP for C: a high-performance library for shared-memory parallel programming. **International Journal of Parallel Programming**, [S.l.], v. 42, n. 4, p. 619–642, 2014.

ZHANG, W.; ZHANG, S.; QI, F.; CAI, M. Self-Organized P2P Approach to Manufacturing Service Discovery for Cross-Enterprise Collaboration. **Systems, Man, and Cybernetics: Systems, IEEE Transactions on**, [S.l.], v. 44, n. 3, p. 263–276, March 2014.

ZHAO, H.; LIU, X.; LI, X. A taxonomy of peer-to-peer desktop grid paradigms. **Cluster Computing**, [S.l.], v. 14, n. 2, p. 129–144, 2011.