UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA
2014.1

BALANCEAMENTO DE *CLUSTER* SEMÂNTICOS POR MEIO DA IDENTIFICAÇÃO DE *HOT-PEERS*

TRABALHO DE GRADUAÇÃO

Aluno: João Paulo Aguilera Borges (jpab@cin.ufpe.br)
Orientador: Ana Carolina Salgado (acs@cin.ufpe.br)

Recife, Julho de 2014

Agradecimentos

Gostaria de agradecer primeiramente a minha família, pois sem o apoio deles não teria iniciado a minha carreira na área de computação e muito menos terminado a graduação. Agradecimento especial ao meu pai João Paulo Borges e minha irmã Tainara Aguilera pelo carinho e apoio nos momentos difíceis. À minha mãe que sempre me apoiou e me incentivou e que continua me motivando mesmo não estando mais neste mundo.

A meus amigos e companheiros que por muitas vezes serviram como família.

Aos meus professores do centro de informática, por compartilharem os conhecimentos e me fazerem evoluir como profissional e como pessoa. À minha orientadora, Doutora Ana Carolina Salgado pela ajuda, incentivo e por me dar a oportunidade de fazer parte do projeto.

Aos colegas de projeto SPEED pelo apoio e ajuda em especial no entendimento do código.

A todos que de forma direta ou indireta contribuirão para minha graduação e realização deste trabalho.

Muito Obrigado!

Resumo:

Peer Data Management Systems (PDMSs) são considerados uma extensão natural de sistemas de integração de dados. Com o diferencial na representação e localização dos dados através da utilização de semântica, o projeto SPEED (*Semantic PEEr-to-Peer Data Management System*) [1] foi proposto. Por se tratar de um sistema distribuído semanticamente apresenta problemas singulares no balanceamento de carga. O objetivo geral desse Trabalho de Graduação é a especificação e implementação de um mecanismo de identificação de *peers* sobrecarregados (*hot-peers*). Esse mecanismo é essencial para as operações de balanceamento no SPEED. O experimento mostrou que o isolamento de *hot-peers* por meio da migração do mesmo para outro *cluster* resulta em um melhor tempo de resposta para as consultas. Este trabalho aborda ainda um algoritmo de balanceamento completo que utiliza a métrica de identificação de *hot-peers* para determinar o melhor balanceamento possível para o sistema. Os algoritmos foram implementados e avaliados como parte integrante do SPEED.

Palavras chave: PDMS, P2P, Web Semântica, Balanceamento de carga

Abstract:

Peer Data Management Systems (PDMSs) are a natural extension for data integration systems. The SPEED (*Semantic PEEr-to-Peer Data Management System*) [1] is a PDMS that uses semantic information to represent and organize peers in the network. The project is semantically distributed and therefore has unique challenges in the load balance realm. The main objective of this dissertation is to implement a mechanism to identify *hot-peers*, i.e. peers that are overloaded. This mechanism is essential for the existing load balance operations in the SPEED system. The experiment has showed that isolating *hot-peer*, by migrating the peer from one cluster to another, results in a better query response time. This dissertation also presents a complete load balance algorithm that uses the hot-peer identification mechanism in order to choose the best load balance operation for the system. All the algorithms was coded and tested as a part of the SPEED project.

Key words: PDMS, P2P, Semantic Web, Load Balance

Sumário

1. Introdução: **8**

2. Fundamentação Teórica **10**

2.1 *Peer-to-Peer* **10**

2.2 PDMS **12**

2.3 Web Semântica **14**

2.4 SPEED **15**

2.5 Balanceamento em redes P2P **17**

2.5.1 Desbalanceamento devido a carga dos nós 17

2.5.2 Critérios de desbalanceamento semânticos 18

2.6 Considerações **18**

3. Balanceamento semântico e de carga no SPEED **20**

3.1 Operações de balanceamento **20**

3.1.1 Migração 21

3.1.2 Divisão 22

3.1.3 União 24

3.2 Considerações **25**

4. *Hot-peers* em *clusters* semânticos **26**

4.1 Algoritmo de identificação do *hot-peer* **26**

4.2 Migração do *hot-peer* **28**

4.3 Algoritmo de balanceamento completo **31**

4.4 Considerações **32**

5. Implementação e análise dos resultados **33**

5.1 Escopo da implementação **33**

5.2 Cenários de testes **34**

5.2.1 Cenário 1: Detecção e migração do *hot-peer*..... 36

5.2.2 Cenário 2: Balanceamento completo de carga 38

5.3 Considerações **42**

6. Conclusão e trabalhos futuros **43**

7. Referências Bibliográficas **45**

Lista de figuras

Figura 1: Anel DHT implementado pelo algoritmo *chord*. Fonte: Souza [7] 12

Figura 2: Arquitetura de um *Peer Data Management System*. Fonte: Calvanese [4] 13

Figura 3: Arquitetura do SPEED. Fonte: Souza [7] 15

Figura 4: Processo de migração no SPEED. Fonte: Silva [29] 22

Figura 5: Macro-etapas do processo de divisão de cluster. Fonte: Souza (Souza, 2012) 23

Figura 6: Divisão de *cluster*. Fonte: Silva [29] 24

Figura 7: Representação do algoritmo de união. Fonte: Silva [29] 25

Figura 8: Algoritmo de detecção de um *hot-peer*. Fonte: O autor 27

Figura 9: Algoritmo de migração do *hot-peer*. Fonte: O autor 30

Figura 10: Diagrama de atividades do algoritmo de balanceamento completo. Fonte:Silva [29] 32

Figura 11: Tempo de execução do algoritmo de detecção de *hot-peer* pelo número de *peers* na rede. Fonte: O Autor 36

Figura 12: Tempo de resposta médio antes e depois da operação de migração do hot-peer. Fonte: O Autor 37

Figura 13: Topologia inicial da rede para o cenário 2. Fonte: O Autor 39

Figura 14: Topologia resultante após algoritmo de balanceamento. Fonte: O Autor 41

Figura 15: Topologia resultante para limiar de carga entre 0.6 e 0.8. Fonte: O autor 41

Lista de tabelas

Tabela 1: Comparativo migração Silva (2013) e *Hot-peer* 37

Tabela 2: Parâmetros de inicialização do cenário 2 40

Tabela 3: Tempo de execução do algoritmo de balanceamento completo 42

1. Introdução:

A internet cresce de uma maneira exponencial e a organização da informação vem se tornando um desafio cada vez maior. A Web Semântica é considerada uma evolução da internet, pois organiza a informação utilizando-se de vocabulários compreensíveis para seres humanos e máquinas. Entretanto, a mineração dos dados e a própria Web Semântica têm muito a evoluir [2].

No âmbito do crescimento da internet, a arquitetura *peer-to-peer* (P2P) vem se destacando. Essa arquitetura está presente em redes de compartilhamento de arquivos, em sistemas de comunicação VOIP (*voice over IP)* e, mais recentemente, como unidade monetária [3]. Esses sistemas são distribuídos, provêem tolerância a falha, eficiência para fontes heterogêneas e por isso têm se popularizado.

Unindo características da Web Semântica e da arquitetura P2P os *Peer Data Management System* (PDMS) são aplicações líderes no compartilhamento de informação semanticamente rica. Nos PDMS cada *peer* é uma fonte autônoma de dados e a organização da arquitetura e a troca de informação entre os *peers* são realizados por meio de esquemas [4]. Esse tipo de sistema traz as vantagens dos sistemas *peer-to-peer* para os gerenciadores de dados, porém podem apresentar problemas na organização da informação e no roteamento das consultas.

O SPEED (*Semantic PEEr-to-peer Data management system*) [1] é um PDMS que tenta solucionar os tradicionais problemas de troca de informação nesse tipo de sistema organizando os *peers* em *clusters* semânticos. Sua arquitetura é mista composta pela interligação de *peers* semânticos ligados por DHTs (*distributed hash table*) e *peer* de integração e de dados que compõem a parte dos *super-peers*.

Todo sistema distribuído tende a sofrer com o desbalanceamento dos recursos da rede. O sistema SPEED não é diferente. Podem haver *peers* que recebam mais requisições que o aceitável levando assim à degradação do desempenho de toda a rede. O SPEED sofre ainda com o desbalanceamento semântico caso a ligação entre os *peers* e os *clusters* esteja comprometendo a integridade semântica da rede. Trabalhos foram propostos para lidar com ambos os tipos de desbalanceamento no sistema SPEED [5,6,7], entretanto as operações de balanceamento

propostas consideravam a existência de um mecanismo de detecção dos nós sobrecarregados, os chamados *hot-peers*.

Esse trabalho visa complementar as operações de balanceamento previamente propostas através da especificação de um algoritmo de detecção dos *hot-peers*. Esse algoritmo se adequa às características do SPEED e é testado em conjunto com as operações já existentes. O trabalho apresenta ainda um método de balanceamento completo para o sistema visando manter a integridade semântica e de carga dos *peers* que compõem o SPEED. Os métodos propostos serão testados e avaliados quanto à performance e à eficiência semântica.

2. Fundamentação Teórica

Essa monografia foi escrita como parte do projeto SPEED (*Semantic PEEr-to-peer Data management system*) [1] e é essencial para o desenvolvimento do trabalho um conhecimento sobre as estruturas que compõem o projeto. Portanto, esse capítulo aborda conceitos como PDMS, Web Semântica, redes DHT e *Cluster* semânticos.

Esse capítulo é composto ainda por uma discussão das diversas maneiras de se realizar um balanceamento em redes *Peer-to-Peer*. Também retrata conceitos básicos como critérios de desbalanceamento e de definição de *hot-peers* .

2.1 *Peer-to-Peer*

Peer-to-peer (P2P) é uma arquitetura para sistemas distribuídos em que cada componente, chamado de *peer*, se comunica de maneira direta e independente sem a necessidade de um controle central (servidor). Essa arquitetura provê tolerância a falhas, dinamismo e eficiência para aplicações que usam fontes de dados heterogêneas. As aplicações que usam essa arquitetura vêm ganhando popularidade nos últimos anos devido ao avanço da Internet e são bem diversas [8]. Há aplicativos de comunicação como Skype, troca de arquivos como o Kazaa e, mais recentemente, até uma moeda virtual Bitcoin utiliza-se do modelo P2P [3].

Os sistemas P2P podem variar quanto ao mecanismo de consulta, o modelo de topologia de rede, o grau de tolerância a falha, entre outras características. Para facilitar o entendimento, os sistemas são geralmente classificados como estruturados, não estruturados ou híbridos. O Quadro 1, adaptada de Stefanidis [9], faz um comparativo entre alguns sistemas P2P.

As redes não estruturadas foram as primeiras a surgir e não possuem uma topologia de rede ou tabela de roteamento definida. Assim, são de fácil implementação, e oferecem vantagens para sistemas com muita entrada e saída de novos *peers*.

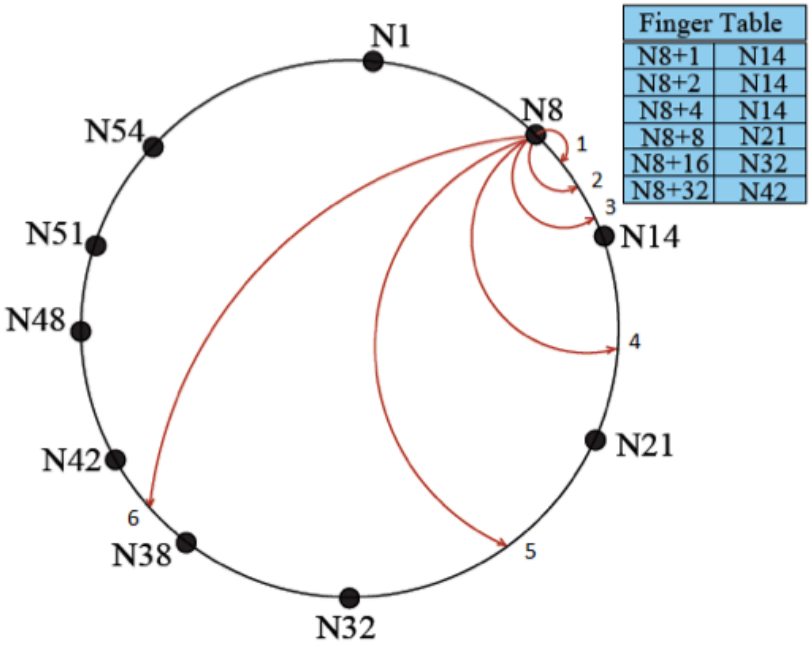
Quadro 1: Um comparativo dos diversos sistemas P2P. Fonte: Adaptado de Stefanidis[9]

Sistema P2P	**Napster**	**Gnutella**	**Chord**	**CAN**
Nível de centralização	Centralizado	Descentralizado	Descentralizado	Descentralizado
Topologia de Rede	-	Desestruturado	Estruturado	Estruturado
Parâmetros	Nenhum	Nenhum	N - Número de nós	N - Número de nós; d –Número de dimensões
Estado roteamento	Constante	Constante	Log(N)	2d
Tamanho do caminho da consulta	O(1)	<TTL	O(logN)	$O(dN^{1/d})$
Tolerância a falha	pobre	pobre	Aleatória	Aleatória

O modelo não estruturado responde as consultas usando os conceitos de inundação (*flowding)* ou caminhos aleatórios (*random walking*), isto é, cada *peer* processa a consulta localmente e depois encaminha a requisição para todos ou alguns dos *peers* vizinhos. O tempo de resposta e a qualidade da informação podem ser um empecilho para as aplicações que usam esse tipo de modelo.

As redes estruturadas possuem uma topologia específica e cada *peer* assume uma determinada porção do conteúdo. Esse modelo requer um maior custo de manutenção, já que cada vez que um *peer* entra ou sai da rede os conteúdos devem ser redistribuídos [8]. As maneiras mais comuns de estruturar uma rede *Peer-to-Peer* são: anéis DHT (Distributed Hash Table) através por exemplo do algoritmo *Chord* [10] como pode ser visto na Figura 1 [7]; espaço de coordenadas cartesianas à exemplo do CAN [11]; árvores balanceadas [12] e grafos *skips* [13].

Figura 1: Anel DHT implementado pelo algoritmo *chord*. Fonte: Souza [7]



2.2 PDMS

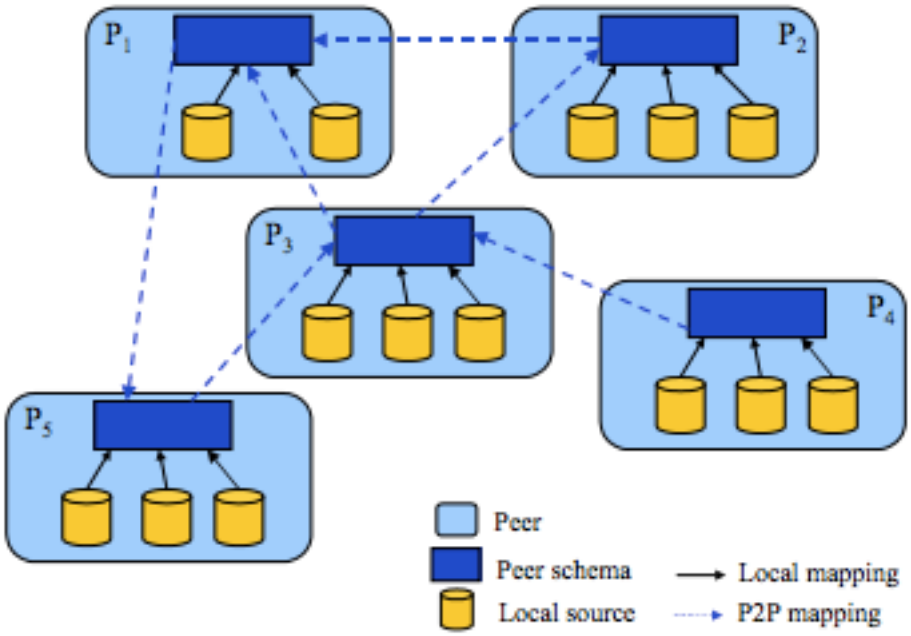
Gerenciar a informação de uma maneira organizada, confiável e de fácil acesso sempre foi um desafio para a computação. As aplicações para esse fim evoluíram dos sistemas centralizados, monolíticos e geralmente ineficientes para um modelo mais dinâmico e descentralizados. Nessa evolução um dos primeiros sistemas a surgir foram os banco de dados distribuídos, que apesar de armazenar a informação em locais distintos, ainda, necessitam de um sistema de controle centralizado e não integram de maneira eficiente sistemas pré-existentes. Os sistemas de bancos de dados federados também possuem características distribuídas, mas permitem que dados de diferentes fontes sejam acessados de maneira transparente por um ponto central. Sistemas de bancos de dados com esquemas globais e sistemas de integração de dados com esquemas de mediação funcionam como um *middleware* provendo transparência para a consulta por meio de um esquema global de dados [14].

Continuando a evolução dos sistemas de banco de dados distribuídos e com o intuito de solucionar os problemas associados à alta demanda dos dados e o aumento no volume do tráfico de informação, os sistemas *Peer Data Management System* (PDMS) foram propostos [4]. PDMS

são uma aplicação para gerenciar dados estruturados em ambientes dinâmicos e heterogêneos típicos das redes P2P. Entretanto, diferentemente de uma aplicação de troca de arquivos tradicionais, os PDMS lidam com dados semanticamente ricos, tornando a estrutura dos *peers* mais complexa.

Nos PDMS cada *peer* funciona como um sistema de integração de dados com esquemas de mediação, ou seja, cada *peer* gerencia um conjunto de dados semanticamente conectados por um mapeamento local (*local mapping*) transformando o conjunto em um esquema global chamado de *peer schema*. Nesse sistema, há ainda um mapeamento chamado de *mapping* que possibilita que os *peers* perguntem e respondam a consultas dos demais *peers* na rede. Os processos de definição dos esquemas devem ser dinâmicos e ágeis e, por isso, são as etapas mais desafiadoras na concepção de um PDMS [1]. A Figura 2[4], exemplifica a arquitetura de um PDMS.

Figura 2: Arquitetura de um *Peer Data Management System*. Fonte: Calvanese [4]



As consultas em um PDMS são muitas vezes reescritas de um *peer* para outro até atingir os *peers* com as informações requisitadas. Esse processo de reescrita, ou enriquecimento da consulta, se não bem executado, pode levar à rápida degradação da informação. Outro fator importante para o processamento é a definição de um limiar de tempo, pois não há garantia que a informação exista no sistema ou que todos os *peers* respondam a tempo. A qualidade do

processamento da consulta é definida seguindo critérios de completude, granularidade e tempo de resposta [15], podendo serem aceitas respostas incompletas, ou uma seleção com as melhores respostas [16] a depender dos requisitos da aplicação.

Devido a sua arquitetura, os PDMS são flexíveis, dinâmicos e capazes de lidar com mudanças frequentes, como entrada e saída de novos *peers* em tempo de execução. Aplicações descentralizadas com alto tráfego de dados como os sistemas para comércio eletrônico, aplicações relacionadas a turismo ou universidades podem ser beneficiadas pelo dinamismo associado aos PDMS, pois o modelo provê resistência a falhas, eficiência e escalabilidade [4].

2.3 Web Semântica

A Web e a Internet evoluíram muito nos últimos anos, entretanto, a mineração e organização da informação sempre foram um desafio. A Web Semântica surgiu como uma evolução da web tradicional trazendo uma camada de entendimento, semântica, às informações disponíveis. Essa camada extra é especificada através de um vocabulário bem definido que pode ser entendido tanto por computadores quanto por seres humanos [2].

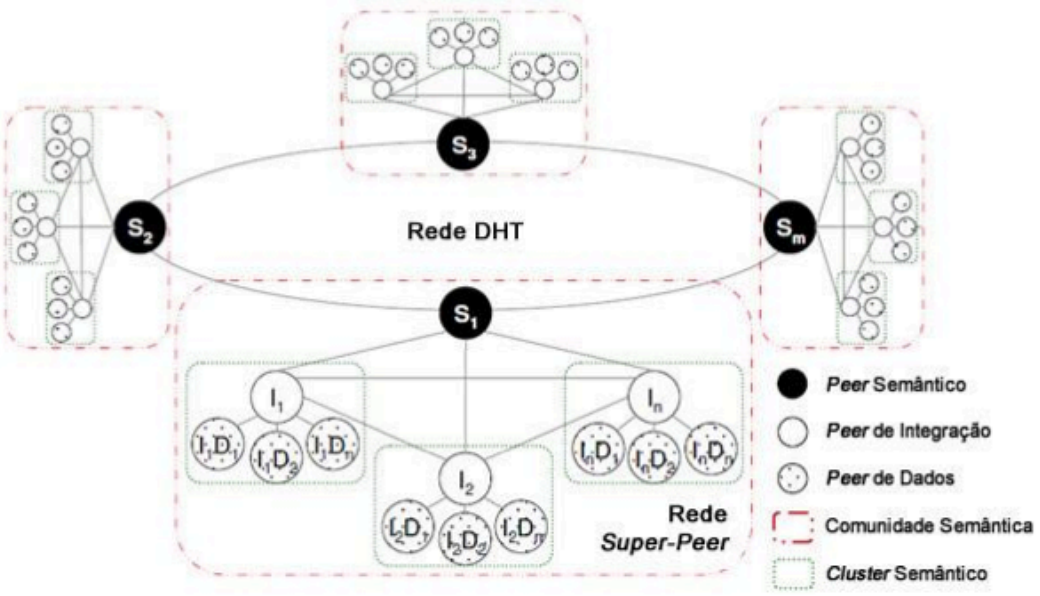
Ontologias são uma especificação de vocabulários para um domínio específico. Esse conceito é parte essencial da Web Semântica. Ontologias são descritas através do uso de classes, objetos, relações e funções. Esse vocabulário é usualmente retratado usando linguagens baseadas em metadados especificadas pela W3C. Uma das primeiras linguagens foi a RDF (*Resource Description Framework*) e funciona como um simples grafo de referência. Uma evolução do RDF foi o OWL(*Web Ontology Language*) que além de organizar a informação ainda apresenta um mecanismo de conferência da congruência dos dados e de seus relacionamentos [17].

A padronização dos dados e das linguagens de descrição de ontologias possibilita que uma estrutura possa evoluir com base em uma ontologia pré-existente. Para saber o grau de similaridade entre ontologias diversas ferramentas de *matching* foram desenvolvidas a exemplo da ferramenta *SemMatcher* [18].

2.4 SPEED

O SPEED (*Semantic PEEr-to-peer Data management system*) é um PDMS que utiliza o conceito de ontologia para prover um serviço de transferência de conhecimento [1]. Possui uma arquitetura baseada em redes DHT (*Distributed Hash Table*) e *super-peers*. A rede *super-peer* do SPEED é organizada em *peers* de dados e *peers* de integração. Esses *peers* são agrupados em *clusters* semânticos. A Figura 3 [7] representa a arquitetura do SPEED.

Figura 3: Arquitetura do SPEED. Fonte: Souza [7]



A primeira camada da estrutura do SPEED são os *peers* semânticos. Esses *peers* devem ser componentes estáveis da rede e guardam as informações sobre a estrutura semântica do sistema. O *peer* semântico utiliza-se de ontologias para possuir uma visão geral da área de conhecimento e das comunidades semânticas, separando assim os *peers* de integração e de dados de acordo com a informação que armazenam.

Os *peer* de integração, ou *super-peers,* são *peers* escolhidos entre os *peers* de dados de um mesmo *cluster* para coordenar as operações. O *peers* de integração (PI) coordenam a entrada e saída dos *peers* de dados do *cluster*, roteiam e enriquecem consultas, elegem *cluster* vizinhos e

repassam as mensagens entre os *clusters* da rede. O *peer* de integração possui a maior carga da rede e o processo de escolha do PI leva em consideração os recursos disponíveis, o *peer* de dados com o maior recurso disponível é escolhido como o líder e promovido a PI.

Os *peer* de dados representam as fontes de dados e são responsáveis por guardar e repassar as informações das consultas. Na arquitetura do SPEED, visto na Figura 3, os *peers* de dados são as extremidades da estrutura, logo não possuem nenhum *peer* conectado a eles.

A formação dos *cluster*s são de acordo com o grau de similaridade dos *peers*. Quando um novo *peer* é inserido na rede a sua semântica é comparada à semântica dos clusters já existentes. Caso haja uma similaridade superior a um limiar chamado de *cluster threshold*, ou limiar de similaridade *intra-cluster*, o peer passa a fazer parte do *cluster* já existente. Caso o novo *peer* não encontre na rede nenhum *cluster* com similaridade superior ao *cluster-threshold* um novo *cluster* é formado.

A troca de informação entre os *clusters* acontece por meio de vizinhanças semânticas. As vizinhanças semânticas são formadas entre os *peer* de integração de acordo com o grau de similaridade entre os clusters. Caso a similaridade entre um *cluster* C_1 e um *cluster* C_2 seja superior ao limiar chamado de *neighbor threshold* ou limiar de similaridade *inter-cluster* a vizinhança entre os *cluster* é formada. O grau de similaridade é medido usando a ferramenta *SemMatcher* [18].

Na arquitetura atual do SPEED os limiares *neighbor threshold* e *cluster-threshold* são valores entre 0 e 1 definidos pelo usuário. Esses valores devem ser definidos antes da inicialização da rede. Os valores usuais são 0.4 e 0.7 para os *neighbor threshold* e *cluster-threshold* respectivamente.

Assim como diversos outros sistemas distribuídos, o SPEED possui ainda um parâmetro para controle do tempo máximo de espera, o TTL (*Time-to-live)*. Esse valor deve ser informado pelo usuário antes da inicialização do sistema e é utilizado para evitar que o sistema espere tempo demais por uma informação.

2.5 Balanceamento em redes P2P

Redes *peer-to-peer* possuem características de independência e heterogeneidade dos nós, tornando problemas de desbalanceamentos inerentes a essa arquitetura. Para identificar se uma determinada organização da rede está balanceada é necessário primeiro estabelecer critérios que definem o desbalanceamento. No SPEED por possuir uma arquitetura baseada em *clusters* semânticos a rede pode estar desbalanceada de acordo com dois grandes critérios: carga dos nós ou dissimilaridade semântica.

2.5.1 Desbalanceamento devido a carga dos nós

No sistema SPEED a carga dos *peers* pode ser oriunda de vários fatores como uma consulta, um roteamento da rede, manutenção da rede, reconfiguração da estrutura, entre outros. Cada fator de carga influencia no percentual de utilização da CPU de cada *peer*. Entretanto, como os *peers* são heterogêneos, a utilização da CPU em isolado pode não representar uma sobrecarga para um determinado *peer*.

O índice de capacidade disponível [19] é um fator que representa melhor a carga para uma rede heterogênea. Esse índice é calculado subtraindo a utilização atual dos recursos do computador da capacidade máxima disponível. A capacidade máxima e a utilização atual podem ser calculadas de diversas maneiras e devem conter informações sobre a CPU, memória, latência de rede, entre outros fatores. Uma maneira de calcular esse índice é através de uma ferramenta de *benchmark* [7].

Após se estabelecer um critério para medir a capacidade computacional, um limiar pode ser utilizado para saber se o *peer* está sobrecarregado [20]. Caso a carga de um determinado *peer* esteja acima do limiar, chamado de *threshold* de carga (TC), o *peer* é considerado desbalanceado.

A carga de um *cluster* é a média das cargas dos *peers* que pertencem ao conglomerado [6]. Para um *cluster* ser considerado desbalanceado, o critério é o mesmo ao utilizado para um *peer* isolado, ou seja, caso a carga do *cluster* esteja acima do *threshold* de carga o *cluster* encontra-se desbalanceado.

2.5.2 Critérios de desbalanceamento semânticos

As ligações dos *cluster* no SPEED são definidas de acordo com a semântica e com os limiares de *cluster threshold* e *neighbor threshold*, entretanto o sistema é dinâmico e a entrada e saída de novos dados pode acarretar alteração no grau de similaridade. Essas alterações podem levar ao desbalanceamento das ligações entre os *peers* do mesmo *cluster,* levando ao desbalanceamento *intra-cluster,* ou entre os *clusters* vizinhos o que acarreta o desbalanceamento *inter-cluster*.

O desbalanceamento *intra-cluster* ocorre quando há pelo menos um *peer* no *cluster* cuja similaridade esteja inferior ao valor do limiar *cluster threhold* [5]. O desbalanceamento *intra-cluster* pode ocorrer quando um novo *peer* entra no *cluster* ou quando a ontologia associada ao *peer* evolui. A saída de um *peer* do *cluster* não causa esse tipo de desbalanceamento, pois ao sair a ontologia só evolui para remover os elementos exclusivos do *peer* que acabou de sair, assim o máximo que pode acontecer é aumentar o grau de similaridade entre os *peers* restantes.

O desbalanceamento *inter-cluster* está associado ao nível de similaridade entre os *cluster*. Esse nível pode ser alterado com a evolução da ontologia de algum dos *cluster*. Caso o nível caia para um valor abaixo do *neighbor threshold* os *clusters* deveriam deixar de ser vizinhos, quando isso não ocorre há um desbalanceamento *inter-cluster*.

Outro tipo de desbalanceamento semântico que pode ocorrer é que *clusters* que não eram vizinhos, ou seja, tinham a similaridade inferior ao limiar *neighbor threshold,* com a evolução das ontologias passem a ter similaridade acima do limiar e, portanto, deveriam se tornar vizinhos. Esse tipo de desbalanceamento não foi explorado por nenhum trabalho anterior relacionado ao SPEED [21,7,5,6] e não será usado como critério de desbalanceamento nesse trabalho.

2.6 Considerações

Neste capítulo foi abordado a base teórica necessária para escrita dessa monografia. Foram discutidos visões gerais sobre: *peer-to-peer,* PDMS, Web Semântica, SPEED e o

balanceamento em redes P2P. O capítulo discutiu ainda sobre os critérios relacionados ao desbalanceamento semântico e de carga e como os assuntos citados no capítulo se conectam.

Complementando o conteúdo sobre balanceamento em redes P2P e conectando ainda mais esse assunto ao projeto SPEED, o próximo capítulo abordará especificamente o balanceamento semântico e de carga no SPEED com uma análise das operações de balanceamento já existentes na literatura.

3. Balanceamento semântico e de carga no SPEED

O SPEED possui características na sua arquitetura que levam a dois tipos de desbalanceamento. A primeira é o desbalanceamento semântico, que ocorre quando os *peers* e suas ligações não estão respeitando os limiares de similaridade definidos. A segunda forma de desbalanceamento está associado à carga da rede e ocorre quando um determinado *cluster* está recebendo a maior parte das requisições, quando a capacidade computacional de um determinado *peer* é inferior a dos demais ou quando o *cluster* está ocioso. Para tratar ambas as causas do desbalanceamento as operações de balanceamento foram estudadas e serão apresentadas nesse capítulo.

3.1 Operações de balanceamento

Na literatura existem diversas estratégias para tratar o balanceamento de carga em redes *peer-to-peer*, todas com dois objetivos principais, a redução da carga por *peer* e a diminuição das operações de migração dos dados [22]. Em redes P2P organizadas em *clusters* as abordagens mais tradicionais para o balanceamento são: replicação dos dados [23], migração dos *peers* [24] , migração dos dados [19] e divisão dos *clusters* [25].

Segundo a definição de Saito [26], replicação de dados consiste em manter múltiplas cópias de uma mesma informação em locais separados. Essa abordagem é muito utilizada para aumentar o desempenho e diminuir a latência dos dados [23]. Entretanto, de acordo com os estudos de Martins [27], problemas associados a manter a integridade dos dados e os custos relacionados à migração dos dados podem tornar esta técnica inviável.

A migração dos *peers* consiste em remover um determinado *peer* de um *cluster* e migrá-lo para outro com menor carga. No estudo conduzido por Sung [28] concluiu-se que esta técnica, apesar de eficiente, pode apresentar problemas comuns às outras abordagens para gerenciamento de dados em redes P2P. O estudo mostra que para a migração funcionar adequadamente é

necessário uma visão geral da situação da rede o que nem sempre está disponível. Outro problema levantado, é a possível falta de um *cluster* disponível e a dificuldade de levantar um limiar exato para se considerar um *peer* sobrecarregado.

Migração dos dados é o processo de remover informações de um determinado *peer* sobrecarregado e inseri-las em um outro *peer* menos carregado independente do fato desse novo *peer* pertencer ou não ao mesmo *cluster*. Assim, como a migração dos *peers,* esse processo necessita ter uma visão geral da rede para obter melhor desempenho [19].

O método de balanceamento por divisão dos *clusters* consiste em separar o *cluster* desbalanceado em dois com menor número de *peers*. Qiao [19] constatou que o número de *peers* em um determinado *cluster* altera sua capacidade de se manter balanceado e sugeriu este processo. Segundo o autor, o processo deve seguir critérios para localização dos dados e identificação da carga a fim de maximizar a probabilidade de ambos os *clusters* resultantes estarem balanceados.

Silva e Souza [6,7,5] analisaram as técnicas para balanceamento de *clusters* descritas acima e propuseram operações de balanceamento que se adequassem às peculiaridades do projeto SPEED. Os trabalhos utilizam meta-informações sobre o grau de semelhança semântica entre os *peers* e os *clusters* e o histórico de carga dos *peers* e visam balancear tanto a carga quanto a semântica dos conglomerados. As operações propostas foram migração de *peer*, divisão e união de *clusters* e serão detalhadas a seguir.

3.1.1 Migração

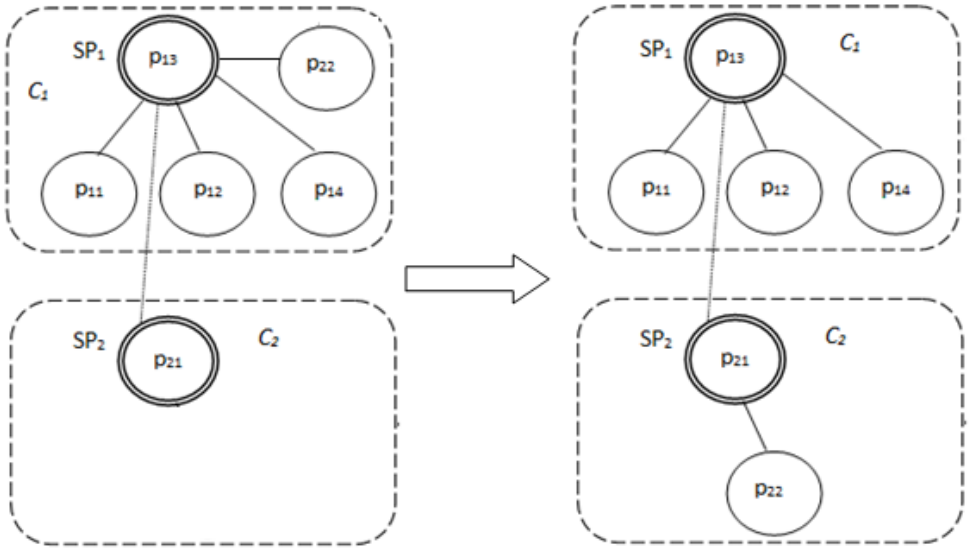
No SPEED, o processo de migração consiste em transferir um *peer* de dados de um determinado *cluster* para algum *cluster* que seja vizinho semântico. O processo pode ser utilizado para obter equilíbrio semântico ou de carga. Caso não haja nenhum vizinho semântico com capacidade computacional para receber o *peer* de dados sendo migrado, um novo *cluster* pode ser formado.

O algoritmo proposto por Souza (2012) elege o *cluster* de destino montando uma lista com todos os vizinhos semânticos do *peer* a ser migrado. O *cluster* com maior similaridade e maior capacidade computacional é então escolhido.

Uma adaptação do algoritmo de migração aceita como parâmetro o *cluster* de destino. Assim, a operação de migração se resume a desacoplar o *peer* de dados, atualizar a ontologia e acoplar ao novo *cluster*. Essa adaptação é utilizada caso já se saiba o *cluster* de destino ou caso o usuário deseje especificar a nova configuração.

A Figura 4 [29] ilustra o processo de migração. Na imagem o *peer* p_{22} está ligado ao *cluster* C_1 cujo *super-peer* é o p_{13}, o processo de migração elege o vizinho semântico C_2 como destino e após o término do algoritmo, o *peer* p_{22} passa a pertencer a este *cluster*.

Figura 4: Processo de migração no SPEED. Fonte: Silva [29]



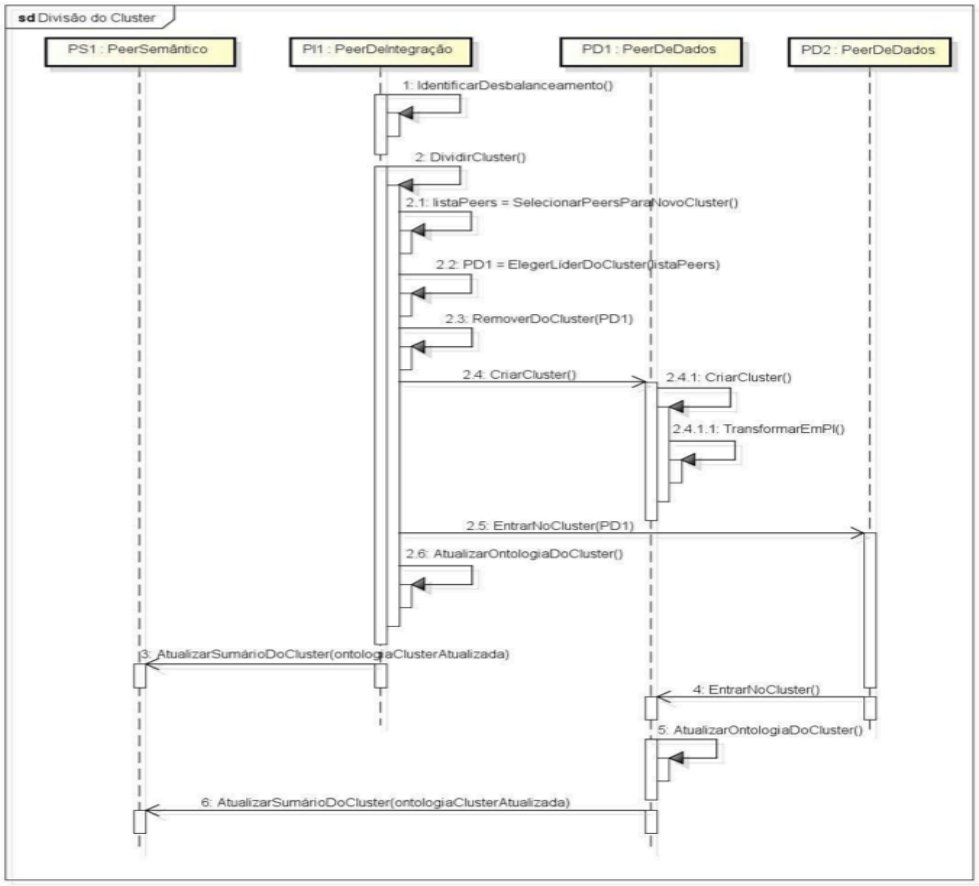
3.1.2 Divisão

A divisão separa os *peers* de dados de um *cluster* em dois. Essa operação é realizada quando a sobrecarga de um *cluster* não é ocasionada pela existência de um *hot-peer* mas sim pelo conjunto de pontos.

Existem duas propostas para a divisão dos pontos no SPEED. A primeira forma, Figura 5 [7], seleciona os dois *peers* menos similares para formar *clusters* distintos. Os demais *peers* são

comparados usando a ferramenta *SemMatcher* aos dois *peers* selecionados, a metade mais similar ao *peer* 1 passa a fazer parte do *cluster* 1, a outra metade passa a fazer parte do *cluster* 2. O líder de cada *cluster* é então definido como o ponto que possui maior capacidade computacional. Os *clusters* 1 e 2 são vizinhos semânticos, logo a consulta que era para ser realizada apenas por um dos *cluster* é executada em ambos, diminuindo o tempo de resposta.

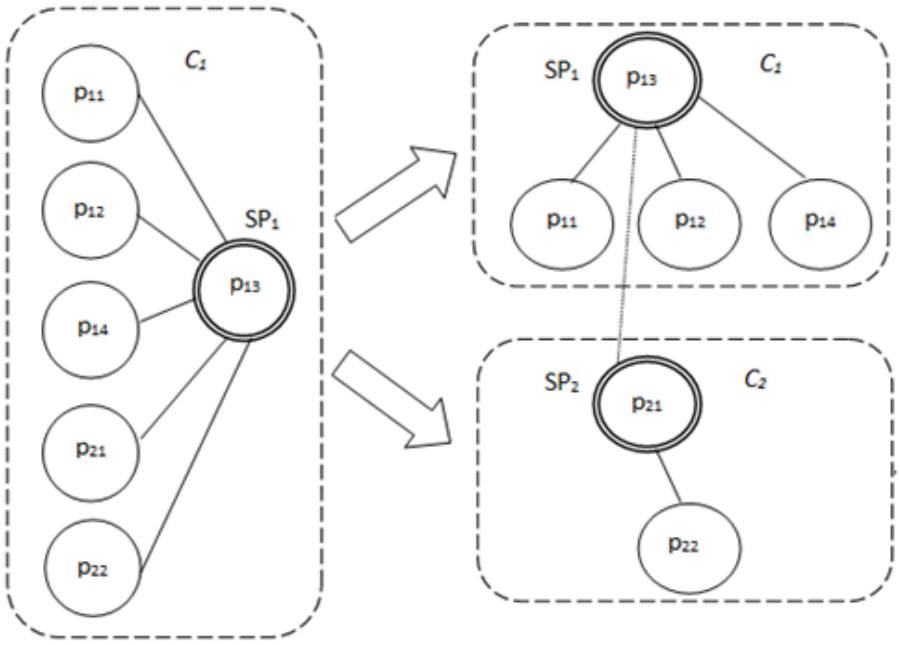
Figura 5: Macro-etapas do processo de divisão de cluster. Fonte: Souza (Souza, 2012)



O processo descrito por Souza (Souza, 2012) , não leva em consideração a evolução da ontologia do *cluster* a medida que novos pontos são inseridos, isso poderia levar ao desbalanceamento semântico ao final da operação. No método de Souza não havia garantia que os dois *clusters* gerados atenderiam ao limiar de similaridade para que fossem vizinhos. O outro método de divisão [5] resolve esse problema, pois funciona de maneira sequencial. No algoritmo o líder do *cluster* a ser dividido, p_{13} Figura 6, é mantido. O líder do novo *cluster* é definido como o *peer* com maior similaridade ao líder do *cluster* original, aumentando a probabilidade de, ao

final da operação, os dois *clusters* resultantes serem de fato vizinhos. Os demais *peers* serão migrados de maneira sequencial para o novo *cluster* caso atendam aos critérios mínimos de similaridades, isto é, a similaridade do *peer* com o *cluster* acima do *cluster threshold*.

Figura 6: Divisão de *cluster*. Fonte: Silva [29]



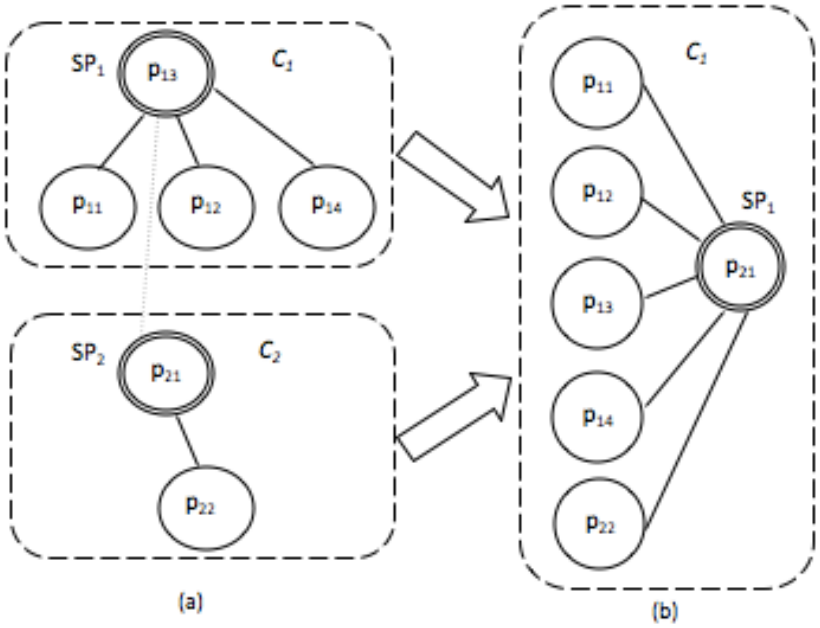
3.1.3 União

O processo de união visa juntar dois *clusters* que estejam com carga inferior a um limiar de carga mínima (TC_{Min}). A união diminui o número de mensagens entre *cluster* vizinhos e, com isso, pode resultar em um melhor desempenho. Espera-se também que após a união haja menos recursos ociosos na rede. Um dos cuidados principais no processo é evitar o desbalanceamento semântico.

O algoritmo de união [6] recebe como parâmetro um *cluster* C_1 e o limiar de similaridade mínimo para junção L_{MIN}. C_1 irá iniciar o processo de busca por algum *cluster* que possua similaridade superior ao limiar L_{MIN}, provavelmente um vizinho de C_1, e que esteja com carga inferior ao limiar TC_{MIN}. Ao encontrar o *cluster* C_2 que preencha esses requisitos o algoritmo compara o número de *peers* em cada *cluster*. O *cluster* que possuir a menor quantidade de *peers* transfere todos os pontos para o outro *cluster*. A Figura 7 [29] ilustra o processo. O lado

esquerdo representa dois *clusters* C_1 e C_2 com similaridade superior ao L_{MIN} e com carga inferior ao limiar TC_{MIN}. O lado direito representa o *cluster* resultante após a união.

Figura 7: Representação do algoritmo de união. Fonte: Silva [29]



3.2 Considerações

Este capítulo abordou as operações de balanceamento para o sistemas *peer-to-peer* baseados em semântica. Apresentou detalhadamente as operações de migração, divisão e união e comparou diferenças nas implementações de Silva [29] e de Souza [7].

No próximo capitulo essas operações serão conectadas para fornecer um melhor balanceamento para a rede. Será apresentado também algoritmos de detecção e migração de *hot-peer* essenciais para a melhor utilização das operações de balanceamento citadas.

4. *Hot-peers* em *clusters* semânticos

As operações de balanceamento citadas no capítulo anterior muitas vezes assumem a existência de um mecanismo de identificação dos *hot-peers,* entretanto não há nenhum algoritmo formalmente proposto para identificá-los que se adeque ao projeto SPEED. Este capítulo irá apresentar um método de identificação de *hot-peer* para *cluster* semânticos e modificações aos algoritmos de balanceamento para que os mesmo passem a utilizar este método proposto.

4.1 Algoritmo de identificação do *hot-peer*

No âmbito de redes P2P baseadas em *clusters* um *hot-peer* é definido como o nó que está sobrecarregado e recebe cargas acima da média do seu *cluster*. Esse nó pode ou não simbolizar um gargalo na rede [22]. Na arquitetura do SPEED a existência de um *hot-peer* ocorre quando um determinado *cluster* está sobrecarregado devido à existência de uma sobrecarga em um único *peer* de dados. Esse *peer* de dados processa requisições mais custosas que a média do cluster e está, portanto, aumentando a média de carga de todo o *cluster*. A Figura 8 descreve o algoritmo de detecção de *hot-peer*.

Na arquitetura do SPEED o primeiro passo para identificação dos *hot-peers* é descobrir quais são os *clusters* sobrecarregados. Souza definiu que para um *cluster* ser considerado sobrecarregado o seu histórico de carga deve ser acima de um limiar chamado de TC_{MAX} (linhas 15 à 35). O histórico de carga é definido usando um *benchmark* que mede informações sobre a disponibilidade dos recursos da máquina como CPU, memória, acesso a disco, entre outras. Na proposta de Souza [7] o *benchmark* deve ser executado a cada intervalo de tempo e os resultados devem ser armazenados em cada *super-peer*. Entretanto, armazenar todo o histórico de carga para cada *super-peer* é uma operação custosa. Por isso, neste trabalho, apenas a média de carga foi armazenada e atualizada seguindo a equação:

$$Media_{nova} = \frac{N_{samples} * Media_{anterior} + Carga_{atual}}{N_{samples} + 1}$$

Após identificar os *clusters* sobrecarregados o próximo passo é definir se a sobrecarga é devido majoritariamente a um único *peer* ou aos diversos *peers*. Para isso, cada *super-peer* deve armazenar a média dos custos de cada requisição recebida e qual foi o *peer* de dados responsável por processá-la (linhas 44 e 45). Em sistemas de integração de dados como o SPEED uma requisição pode ser interpretada como uma consulta cujo custo pode ser estimado usando técnicas estatísticas [30]. Obermeier e Nixon propõem um algoritmo para medir o custo de uma consulta *sparql* baseado em suas operações como junção e seleção [31]. Como as requisições no SPEED podem ser definidas como consultas *sparql* [32], esta métrica foi adotada.

Cada *peer* de dados deve armazenar o número de vezes que respondeu uma requisição custosa (N_{QC}). Uma requisição é considerada custosa se o seu custo, definido usando a métrica proposta por Obermeier, for superior à média dos custos armazenados no *super-peer*. Usando a informação dos N_{QC} de cada *peer* de dados a probabilidade de um determinado *peer* ser um *hot-peer* é calculada (linha 50) usando a seguinte formula:

$$P_{hotpeerI} = \frac{N_{QCI}}{\sum_{j=0}^{\#\ de\ PDs\ no\ cluster} N_{QCj}}.$$

Caso essa probabilidade seja acima de 0.5, o *peer* é considerado um *hot-peer* (linha 52).

Figura 8: Algoritmo de detecção de um *hot-peer*. Fonte: O autor

```
1.     Algoritmo Deteccao_HotPeer(TCMax:Real)
2.     {
3.        S_sobrecarregados, SV, SV2: Conjunto de Clusters;
4.        p: Ponto;
5.        SPi: Conjunto de Pontos;
6.        Ck, Cj, Cs: Cluster;
7.        HCmedia_ci,   HCmedia_ck,   HCmedia_cj,   NQCi,   NQCTotal,
   PHotPeer, Np : Real;
8.        HCdestino: HC;
9.        SClusterPonto: Conjunto com o pares Cluster-Ponto;
10.
11.       Ci ← recuperar cluster corrente;
12.       HCmedia_ci ← obter a média do HC de Ci;
13.       SV ← recuperar os vizinhos semânticos de Ci;
14.
15.       SE (HCmedia_ci >= TCMax) ENTÃO {
16.           Inserir Ci em S_sobrecarregados;
17.       }
18.
19.       PARA CADA Ck EM SV FAÇA {
```

```
20.          HCmedia_ck ← obter a média do HC de Ck;
21.          //Verificando se o Ck está sobrecarregado.
22.          SE (HCmedia_ck > TCMax) ENTÃO {
23.             Inserir Ck em S_sobrecarregados;
24.          }
25.
26.         SV2 ← recuperar os vizinhos semânticos de Ck;
27.
28.         PARA CADA Cj EM SV2 FAÇA {
29.            HCmedia_cj ← obter a média do HC de Cj;
30.            SE (HCmedia_cj > TCMax) ENTÃO {
31.                SE (Cj ∉ S_sobrecarregados) ENTÃO{
32.                   Inserir Cj em S_sobrecarregados;
33.                }
34.           }
35.         }
36.
37.        PARA CADA Cs EM S_sobrecarregados FAÇA {
38.            SPi ← Conjunto de pontos de Cs;
39.            NQCTotal ← 0;
40.            Np ← Número de pontos em Cs;
41.
42.            SE (Np > 1) ENTÃO{
43.              PARA CADA p EM SPi FAÇA {
44.                 NQCi ← Numero de querys custosas em p;
45.                 NQCTotal ← NQCTotal +  NQCi;
46.              }
47.
48.              PARA CADA p EM SPi FAÇA {
49.                 NQCi ← Numero de querys custosas em p;
50.                 PHotPeer ← NQCi / NQCTotal;
51.
52.                 SE (PHotPeer > 0.5) ENTÃO{ //É hot-peer
53.                    Inserir <Cs,p> em SClusterPonto;
54.                 }
55.              }
56.           }
57.         }
58.
59.         RETORNAR SClusterPonto;
60.    }
```

4.2 Migração do *hot-peer*

Em sistemas P2P baseados em *clusters* a migração dos *peers* sobrecarregados é uma tarefa essencial, porém complicada, pois deve-se encontrar quais são os *peers* que devem ser migrados e o melhor destino para esses *peers* para que se tenha o equilíbrio adequado. Para o SPEED, o desafio é ainda maior, pois o balanceamento semântico também deve ser mantido.

A Figura 9 ilustra um possível algoritmo para migração do *hot-peers* no sistema SPEED. A operação descrita visa prover balanceamento semântico e de carga para o sistema. O algoritmo utiliza-se dos métodos de detecção de *hot-peers* (Figura 8), e do método de migração de *peer* (Figura 5) [7].

A primeira etapa do algoritmo é identificar se o *peer* (p) que está tentando realizar a migração é de fato um *hot-peer*. Para isso, o algoritmo de detecção de *hot-peer* é executado e uma lista contendo todos os *hot-peers* do sistema é gerada (linha 10). Caso o elemento *p* pertença à lista, então o processo de migração continua.

O processo para escolha do destino de um determinado *peer* sobrecarregado é através da criação de uma lista de *clusters* candidatos [6]. Os *clusters* candidatos podem receber o *peer* a ser migrado sem alterar o balanceamento semântico do sistema, ou seja, a similaridade entre o *peer p* o *cluster* destino é maior que o limiar *cluster threshold* passado como parâmetro do sistema (linha 25).

Caso haja mais de um *cluster* candidato, a lista deve ainda ser organizada em ordem crescente de capacidade computacional, assim o *cluster* candidato com maior capacidade computacional disponível é escolhido para a migração (linha 34). Caso só haja apenas um *cluster* candidato esse também será o *cluster* de destino (linha 37). Após a definição do *cluster* de destino, para ambos os casos, o processo de migração segue o algoritmo de migração tradicional [7]. Ou seja, o *peer* deixará o *cluster* atual e atualizará a ontologia deste *cluster* removendo os conceitos que são únicos a este *peer* por um processo de *unmerging* e entrará no novo *cluster* por um processo de *merging* evoluindo a ontologia deste novo *cluster* (linha 45).

Na eventualidade de não encontrar nenhum *cluster* para migrar o ponto, um novo *cluster* deve ser formado (linha 40). Esse novo *cluster* terá o *peer p* como *super-peer* da rede e será vizinho do antigo *cluster* (linha 41).

Figura 9: Algoritmo de migração do *hot-peer*. Fonte: O autor

```
1.    Algoritmo Migração_HotPeer(p:Ponto; θ, TCMax:Real)
2.    {
3.       SV_candidatos, SV_i : Conjunto de clusters;
4.       SP_i:Conjunto de pontos;
5.       C_destino,C_i : Cluster;
6.       HC_media_v, NCandidatos: Real;
7.       SClusterPonto: Conjunto com os pares Cluster-Ponto;
8.
9.       Ci ← recuperar o cluster associado a p;
10.      SClusterPonto ← Deteccao_HotPeer(TCMax);//Lista com os hot-
      peers
11.      SE (p ∈ SCluserPonto) ENTÃO { //p é um hot-peer
12.         SV_i ← recuperar os vizinhos C_i;
13.
14.         //Buscando um cluster de destino para o ponto
15.         SV_candidatos ← ∅;
16.         //Buscando clusters candidatos para o ponto
17.         PARA CADA v EM SV_i FAÇA {
18.            SE v não está em manutenção ENTÃO {
19.               HC_media_v ← recuperar média de HC de v;
20.
21.               //Verificando se o cluster está sobrecarregado
22.               SE (HC_media_v < TCMax) ENTÃO {
23.                  //Verificando se o ponto possui similaridade
24.                  //semântica mínima com o cluster
25.                  SE (GSM(O_p, O_v)≥ θ) ENTÃO
26.                      inserir v em SV_candidatos;
27.               }
28.            }
29.         }
30.         NCandidatos ← Número de elementos em SV_candidatos;
31.
32.         SE (NCandidatos > 1)ENTÃO {
33.               Ordernar SV_candidatos em ordem crescente de HC;
34.               C_destino ← SV_candidatos[0];
35.         }
36.         SE(NCandidatos=1) ENTÃO// Só um cluster foi encontrado
37.            C_destino ← SV_candidatos[0];
38.         SENÃO { //Se não encontrado um cluster destino,
39.                 //formar novo cluster com o ponto.
40.            C_destino ← formar novo cluster com p;
41.            estabelecer vizinhança entre C_i e C_destino;
42.            RETORNAR;
43.         }
44.        }
45.        Migrar(p,C_destino);//Processo de migração de peer
46.       }
47.   }
```

4.3 Algoritmo de balanceamento completo

Silva [29] propôs um algoritmo baseado nas operações de balanceamento descritas no Capítulo 3. Esse algoritmo, chamado de algoritmo de balanceamento completo, deve ser executado em cada *super-peer* e realiza as operações de balanceamento adequadas para cada tipo de situação. O algoritmo visa manter os critérios semânticos e de carga balanceados e depende do algoritmo de migração do *hot-peer* (Figura 9).

Cada *super-peer* deve executar o algoritmo proposto de tempos em tempos, a fim de garantir o balanceamento. A proposta é que, após todos os *peers* terem executado o algoritmo localmente, a rede, como um todo, esteja balanceada. O fluxo para o balanceamento pode ser visto na Figura 10.

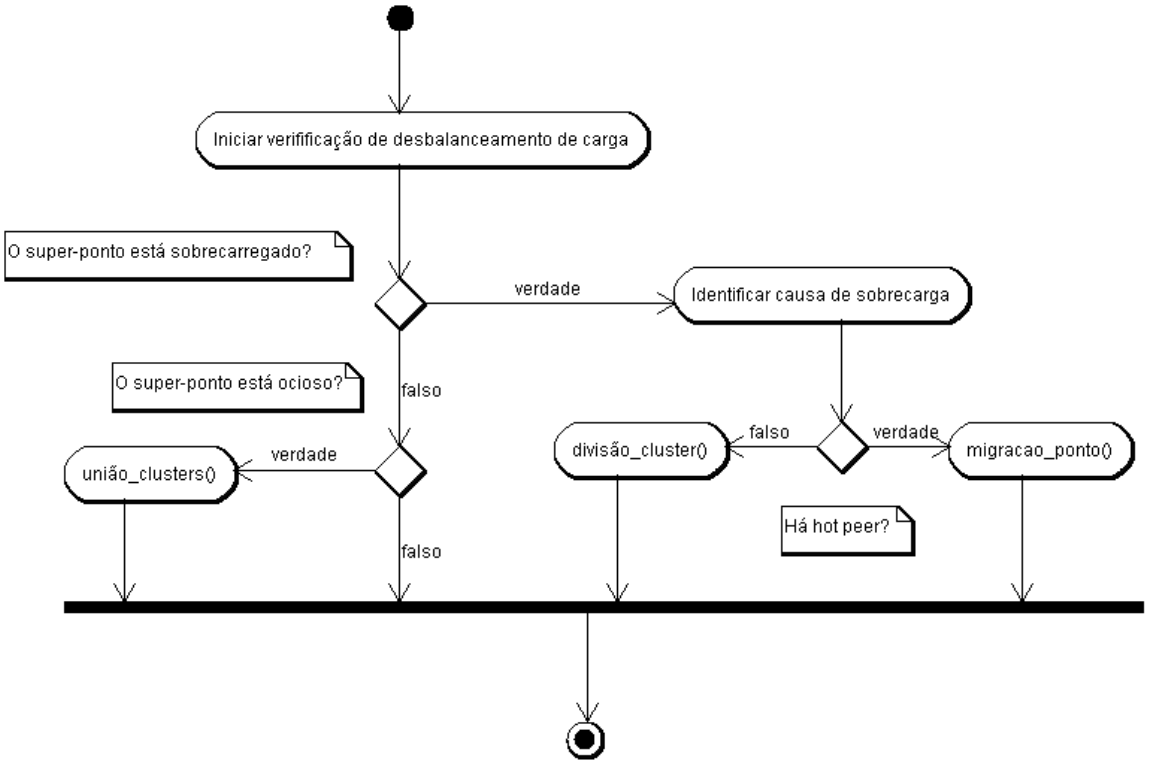
A primeira etapa do algoritmo é identificar se o *super-peer* está com carga acima do limite. Para isso, a média do histórico de carga do *super-peer* é comparada com o limiar de carga máxima (TC_{max}), se o valor da média for superior ao do limiar o *cluster* está desbalanceado e deve, portanto, continuar a execução do algoritmo de balanceamento.

A próxima etapa é identificar se a causa do excesso de carga no *super-peer* é devido à existência de um *hot-peer* e tentar migrá-lo. Essa etapa utiliza-se do algoritmo de migração dos *hot-peers* (Figura 9), ou seja, caso seja detectada a presença de um *hot-peer* o algoritmo tentará achar um *cluster* candidato para migrá-lo e caso não consiga, um novo *cluster* deve ser gerado.

Caso na etapa anterior se constate que não há um *hot-peer* ligado ao *super-peer* então uma operação de divisão deve ser realizada. A operação de divisão segue o algoritmo tradicional de divisão, apresentado na Seção 3.1.2.

O algoritmo de balanceamento completo ainda lida com situação de ociosidade. Para identificar se um cluster está ocioso, o histórico de carga do *super-peer* é comparado a um parâmetro definido pelo usuário chamado de limiar de carga mínima (TC_{MIN}). Se a média do histórico de carga do *super-peer* for inferior a este limiar, então o *cluster* é considerado ocioso e a operação de união, Seção 3.1.3, é realizada.

Figura 10: Diagrama de atividades do algoritmo de balanceamento completo. Fonte: Silva [29]



4.4 Considerações

Esse capítulo apresentou um método de identificação de *hot-peer* que se adequa as características do SPEED e utilizou esse método para implementar algoritmos de migração de *hot-peer* e de balanceamento completo.

O capítulo seguinte apresentará a implementação e teste desses algoritmos e analisará os resultados.

5. Implementação e análise dos resultados

O SPEED (*Semantic PEEr-to-Peer Data Management System*) vem sendo desenvolvido como parte de um projeto que visa adquirir conhecimentos sobre os sistemas de gerenciamento de banco de dados em ambientes P2P com uma ênfase na semântica [14]. Ao longo dos anos o projeto foi evoluindo incluindo serviços de roteamento de consulta, interface gráfica e balanceamento de carga [33,21]. Este trabalho visa ajudar na evolução do sistema SPEED e, portanto, os algoritmos propostos foram implementados como um módulo do sistema pré-existente.

O sistema SPEED vem sendo desenvolvido na linguagem *Java SE* [34] usando como IDE (*Integrated Development Environment*) a versão clássica do *Eclipse* para *Java*: o *Eclipse Lunar* [35].

Cada *peer* do sistema é um processo isolado que se comunica com os demais *peers* através de mensagens assíncronas enviadas via *socket*. A utilização dessa metodologia de comunicação permite uma melhor utilização dos diferentes recursos que integram a rede, já que os *peers* não aguardam o processamento de seus *peers* vizinhos. Entretanto, a falta do sincronismo apresenta um desafio maior nas fases de desenvolvimento e testes.

5.1 Escopo da implementação

Ao longo deste trabalho várias funcionalidades foram adicionadas ao SPEED. Fizeram parte do escopo de implementação não apenas as funções diretamente envolvidas na confecção dessa dissertação, mas também modificações nas funções de balanceamento de carga propostas por Souza [7] para que tais funções também atendessem aos requisitos semânticos propostos por Silva [5]. Logo, este trabalho incluiu a modificação ou a implementação das seguintes funções:

1. Seleção de *cluster* candidato para migração de *peer* de dados;

2. Identificação do custo da consulta para formulação do histórico de carga;

3.Otimização do histórico de carga para requerer menos memória e tempo de processamento;

4. Parametrização de capacidade operacional do *peer* de acordo com a métrica de custo da consulta;

5. Implementação do mecanismo de identificação de *hot-peers;*

6. Implementação do algoritmo de balanceamento completo proposto por Silva [5] usando o mecanismo de identificação de *hot-peers* proposto neste trabalho.

Souza [7] sugeriu que o intervalo de tempo considerado pela simulação do processamento das consultas nos *peers* de dados e nos *peers* de integração fosse obtido pela divisão do custo de processamento da consulta pela capacidade computacional disponível. A técnica proposta visava simular diferentes *peers* com diferentes capacidades computacionais e diferentes cargas em apenas um computador. Entretanto, para os experimentos propostos, essa métrica não seria tão eficiente. Neste trabalho, o tempo de retorno da consulta é medido como a diferença de tempo entre enviar a mensagem com a consulta e receber a resposta com os resultados.

Para emular as diferentes capacidades computacionais, este trabalho propõe que cada *peer* execute, além da consulta, uma função cujo tempo de resposta esteja associado ao porcentual de utilização do sistema. Essa função deixa a *thread* principal do *peer* dormindo por 100*porcentual milissegundos, não interferindo no tempo de resposta de outro *peer* executando no mesmo computador. Assim, os *peers* que estão com a porcentagem de utilização do sistema alto dormem por mais tempo e demoram mais a responder. Para os experimentos, as capacidades computacionais de cada *peer* foram configuradas manualmente para atender aos cenários de testes.

5.2 Cenários de testes

Os testes foram propostos para explorar diferentes cenários de desbalanceamento em diferentes parametrizações. O tempo de resposta é uma métrica comum em artigos de

balanceamento de carga [19,7]. Portanto, em cada experimento foi medido o tempo para consultas antes e depois do balanceamento mostrando a eficácia dos métodos propostos. Os experimentos mediram também o tempo necessário para que a função de balanceamento fosse completada a fim de mensurar o custo extra associado a cada operação.

Para a realização do experimento, em cada cenário, as seguintes etapas foram executadas:

1. Configuração dos parâmetros de meta-configuração: Limiar de carga máxima (TC_{MAX}), Limiar de carga mínima (TC_{MIN}), tempo limite para consulta (TTL) e porcentagem de utilização da capacidade computacional de cada *peer*;

2. Inicialização e estabilização dos *peers*, seguindo a ordem necessária para a simulação do cenário desejado;

3. Inicialização da plataforma de consulta [33];

4. Realização de cinco consultas distintas e medição do tempo médio de resposta;

5. Execução da função de balanceamento desejada para o cenário e medição do tempo para a estabilização da rede;

6. Realização de cinco consultas e medição do tempo médio de resposta.

Os experimentos foram realizados executando todos os *peers* na mesma máquina. Portanto, o uso da memória cache para consultas de um mesmo objeto poderia ocasionar uma alteração dos resultados. Para evitar esse efeito todas as consultas foram realizadas buscando objetos distintos mantendo o mesmo custo da consulta, seguindo o conceito de custo proposto por Obermeier [31].

O SPEED como proposto por Pires [1] possui uma característica dinâmica e a estrutura da rede pode evoluir alterando assim o grau de similaridade semântica entre os *peers*. Nos experimentos, a matriz de similaridade semântica entre os *peers* também foi analisada afim de averiguar as alterações semânticas provocadas pelas operações propostas.

A máquina que foi utilizada para realizar os experimentos possui as seguintes características:

Processador: 1.8GHz intel core i5
Memória: 4GB 1600MHz DDR3
S.O.: Mac OSX versão 10.9.2
HD: 120 GB SSD
Java: java versão "1.6.0_65"
Java(TM) SE Runtime Environment (build 1.6.0_65-b14-462-11M4609)
Java HotSpot(TM) 64-Bit Server VM (build 20.65-b04-462, mixed mode)

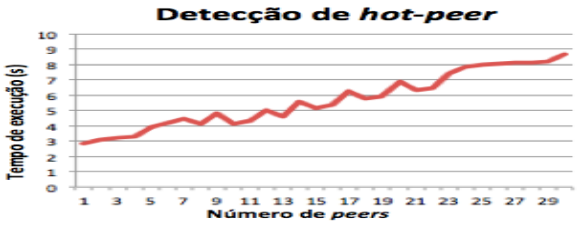
5.2.1 Cenário 1: Detecção e migração do *hot-peer*.

Neste cenário, o tempo para detecção e migração do *hot-peer* foram medidos com números de *peers* no sistema variando de 1 até 30. Os limiares de similaridades *cluster threshold* e *neighbor threshold* foram de 0.5 e 0.4, respectivamente. Além do tempo de execução do algoritmo, esse experimento mostrou a alteração no tempo médio de consulta. Para todos os cenários, mesmo com diversos *peers*, havia apenas um *hot-peer* e, portanto, apenas uma operação de migração foi executada. A eficiência semântica do *cluster* também foi medida, o critério avaliado foi o fator $\bar{\kappa}$ [36]. Esse fator é a razão entre o número de *peers* que está alcançável a um tempo menor que o TTL definido e que o fator de similaridade é superior a um limiar *cluster threshold*, e o número total de *peers* que tem similaridade superior ao *cluster threshold*.

A Figura 11 ilustra a média das medições dos tempos para execução do algoritmo de detecção e migração de *hot-peer* de acordo com a quantidade de *peers* na rede *super-peer* do *SPEED*.

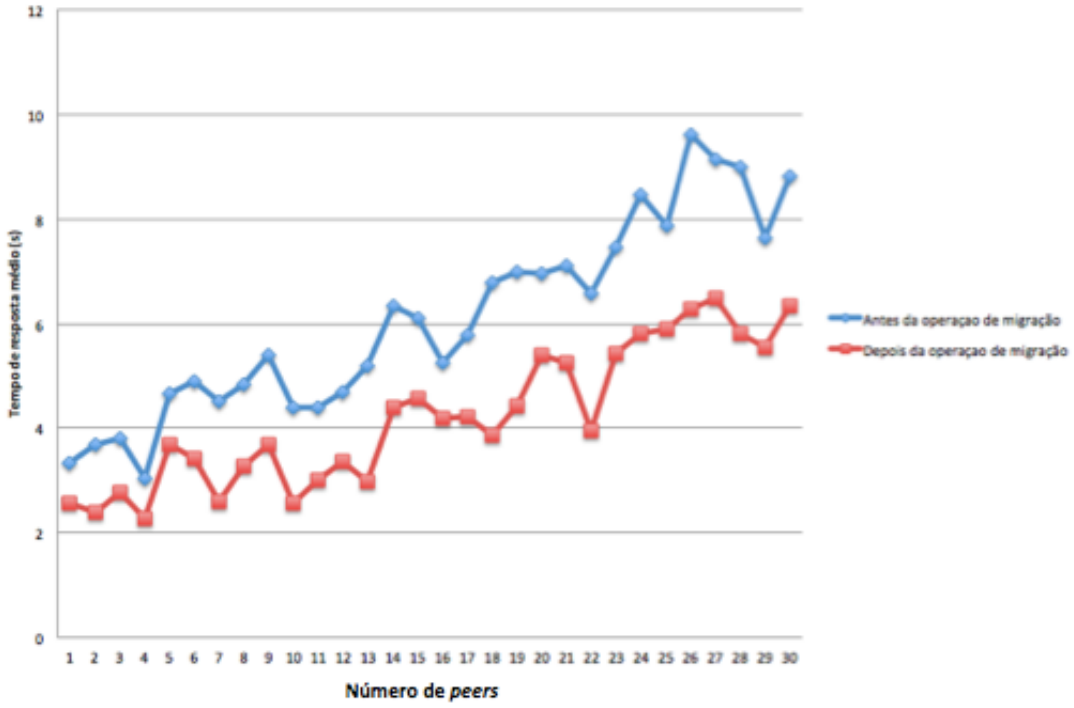
Observa-se que há uma relação quase que linear entre o número de *peers* e o tempo de resposta. Essa relação é esperada, pois o número de mensagens trocadas na rede para descobrir os *hot-peers* varia linearmente com o número de *peers* no sistema.

Figura 11: Tempo de execução do algoritmo de detecção de *hot-peer* pelo número de *peers* na rede.
Fonte: O Autor



A média do tempo de resposta das consultas de acordo com a quantidade de *peers* antes e depois da migração do *hot-peer* pode ser vista na Figura 12.

Figura 12: Tempo de resposta médio antes e depois da operação de migração do hot-peer. Fonte: O Autor



Silva [6] encontrou uma melhora média de 23,99% após a operação de migração de um *peer* aleatório do *Cluster*. Este trabalho, encontrou uma melhoria média de 33,30% no tempo de consulta. O resultado superior é devido à detecção do *hot-peer* e do *cluster* destino que otimiza o processo de busca. O detalhamento do resultado pode ser encontrado na Tabela 1.

Tabela 1: Comparativo migração Silva (2013) e *Hot-peer*. Fonte: O autor

	Sem a operação	Com a operação	Melhora
Migração proposta por [6]	**6,35**	**4,83**	23,99%
Migração do hot-peer	**5,79**	**3,84**	33,30%

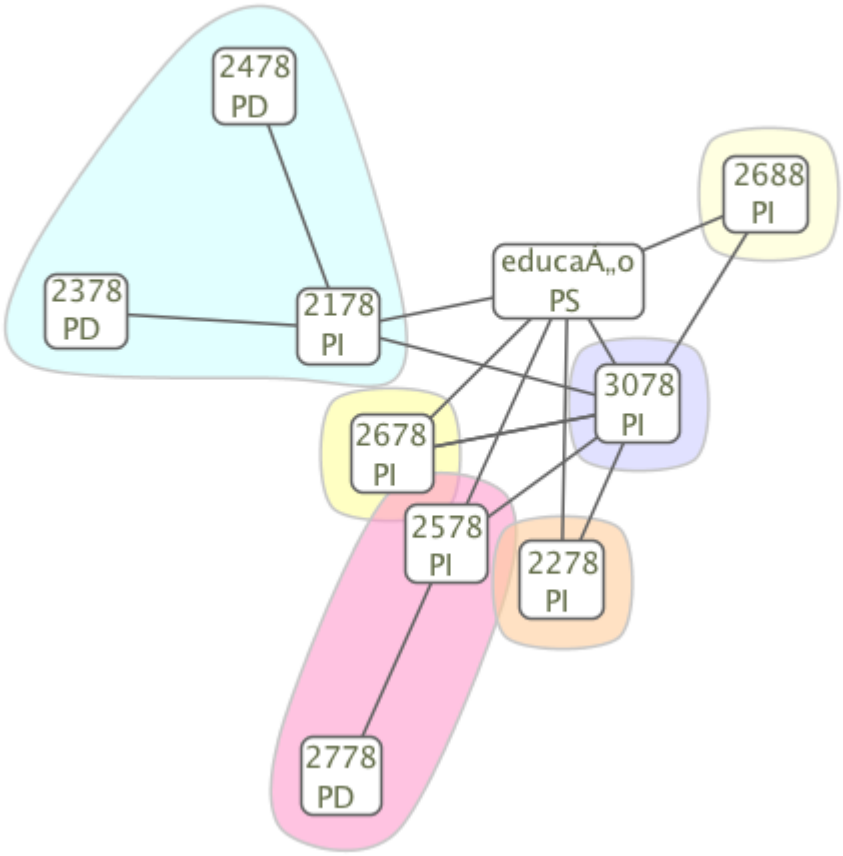
No experimento também foram analisadas as mudanças semânticas provocadas pela operação. Para isso, foi utilizada a função *SemMatcher* [18].O experimento mostrou que a operação de migração do *hot-peer* provoca pouca variação na estrutura semântica da rede. A média da variância do grau de similaridade dos *peers* com o avançar da rede e com a operação de migração foi de apenas 1%. O cenário que produziu a maior variação foi com 20 *peers* onde houve um aumento de similaridade de 24% no nível de similaridade de um *peer* específico e um outro *peer* vizinho.

Antes do experimento a rede se encontrava balanceada semanticamente, isto é, cada *peer* ligado ao *cluster* possui similaridade semântica acima da definida pelo limiar de similaridades *intra-cluster* (*cluster threshold*) e cada *cluster* vizinho ao *peer* possuía similaridade acima do limiar *inter-cluster* (*neighbor threshold*). Assim, a eficiência semântica $\bar{\kappa}$ era 1 e estava maximizada. Devido à pouca variação na semântica, em todos os testes o balanceamento semântico foi mantido e a eficiência semântica foi $\bar{\kappa} = 1$. O resultado de estabilidade semântica era esperado pois o algoritmo leva em consideração fatores de similaridade semântica na escolha do *cluster* de destino do *hot-peer*, como explicado na descrição do algoritmo.

5.2.2 Cenário 2: Balanceamento completo de carga

O cenário 2 foi proposto inicializando 9 *peers* em sequência para medir o tempo para o balanceamento total do sistema. A rede foi iniciada com dois *clusters* sobrecarregados. Um *cluster* devido à existência do *hot-peer* e outro não. A ontologia utilizada foi a mesma proposta por [7] *Education.owl*. Os limiares de similaridades *cluster threshold* e *neighbor threshold* foram de 0.5 e 0.4, respectivamente. A Figura 13 mostra a configuração inicial do sistema para o cenário de teste proposto.

Figura 13: Topologia inicial da rede para o cenário 2. Fonte: O Autor



Para averiguar a eficiência semântica dos agrupamentos, [5] propôs o coeficiente de balanceamento semântico. Para esse experimente, antes da realização da função de balanceamento, o coeficiente de balanceamento semântico era de 0.7.

Para a realização do experimento, os *peers* foram pré-inicializados com uma carga. Os valores dos parâmetros para os limiares e para as cargas de cada *peer* encontram-se na Tabela 2.

Neste cenário o valor do limiar de carga máximo foi variado e o algoritmo de balanceamento de detecção de *hot-peer* foi utilizado. A operação de balanceamento executada e a topologia resultante do sistema varia conforme o limiar de carga máxima (TC_{MAX}).

Tabela 2: Parâmetros de inicialização do cenário 2. Fonte: O autor

Parâmetro	**Valor**
Limiar de carga mínima (TC_{MIN})	0.0
Tempo limite para a consulta (TTL)	100s
% de carga PI 2178	0.8
% de carga PD 2378 e 2478	0.8
% de carga PI 2578	0.6
% de carga PD 2778	0.9
% de carga PI 2278, 2678, 3078 e 2688	0.3

Para valores de limiar de carga máxima inferior a 0.6 ($TC_{MAX} < 0.6$), o algoritmo de balanceamento executou duas funções: migrou o *hot-peer* do PI 2578 e dividiu o PI2178 resultando na topologia da Figura 14.

Quando o limiar de carga máxima estava entre 0.6 e 0.8 ($0.6 \leq TC_{MAX} \leq 0.8$)., o algoritmo de balanceamento executou apenas a função de divisão. A topologia resultante é ilustrada na Figura 15. Para valores de limiar de carga máxima acima de 0.8 o sistema não foi alterado ($TC_{MAX} > 0.8$).

Os tempos de resposta e de processamento da função de balanceamento para cada limiar de carga máxima, assim como o coeficiente de balanceamento semântico após a operação de balanceamento se encontra na Tabela 3.

Pode-se perceber que em quase todos os casos houve uma melhoria do desempenho no tempo de resposta consulta. Além disso, o coeficiente de similaridade semântica aumentou em quase todos cenários. O último caso, com limiar de carga igual a 1 apresentou uma piora no tempo de resposta da consulta, entretanto esse valor é insignificativo, pois para este limiar todos os *peers* estavam com carga inferior ao limiar de carga proposto não acarretando assim alteração na topologia resultante. A piora no tempo de resposta é devida apenas a fatores normais do computador, alheio ao sistema.

Figura 14: Topologia resultante após algoritmo de balanceamento. Fonte: O Autor

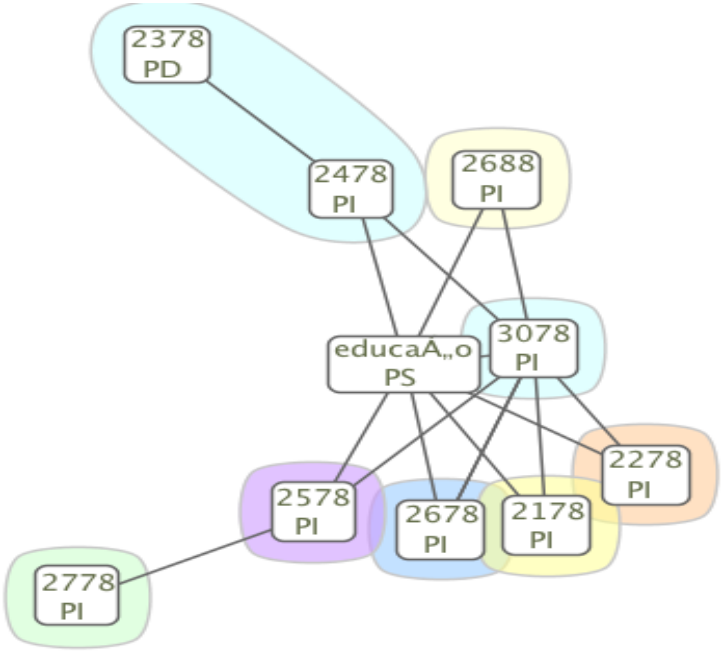


Figura 15: Topologia resultante para limiar de carga entre 0.6 e 0.8. Fonte: O autor

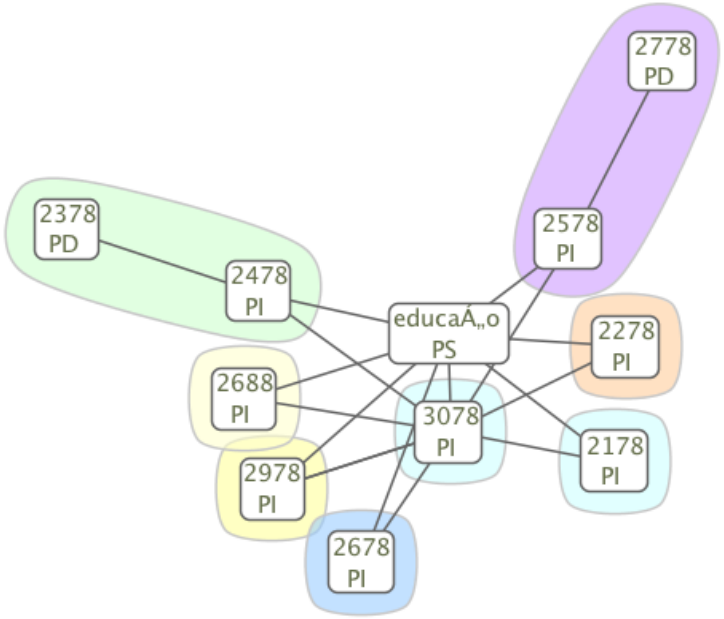


Tabela 3: Tempo de execução do algoritmo de balanceamento completo. Fonte: O Autor

Limiar de Carga máxima	Tempo Balancear (seg)	Tempo resposta da consulta (antes) (seg)	Tempo resposta da consulta (depois) (seg)	Melhora % tempo de resposta da consulta	Coeficiente de balancemanento semantico (Depois)
0.1	8.05	6.754	6.65	1.5398	1
0.2	8.04	6.964	6.38	8.386	1
0.3	8.51	7.743	7.13	7.9168	1
0.4	8.79	7.869	6.99	11.1704	1
0.5	8.54	6.97	6.58	5.5954	1
0.6	6.04	7.967	6.39	19.7942	0.9
0.7	6.346	8.03	5.787	27.9328	0.9
0.8	4.035	7.933	6.399	19.3369	0.9
0.9	4.25	7.852	7.662	2.4198	0.7
1	4.175	7.957	7.968	-0.1382	0.7
Média:	**6.6776**	**7.6039**	**6.7936**	**10.39539**	**0.91**

O tempo para o balanceamento é de no mínimo 4.035s. Esse é o tempo que o sistema leva para detectar as sobrecargas e a natureza do desbalanceamento quando iniciado com 9 *peers*. Como o tempo para o balanceamento completo é superior à melhoria média do tempo da consulta esse processo deve ser utilizado com cautela. Entretanto, caso muitas consultas sejam necessárias ou a rede se encontre muito desbalanceada o procedimento é vantajoso.

De acordo com o experimento, o limiar de carga máxima que apresentou o melhor tempo de resposta e com um custo de procedimento médio está entre 0.6 e 0.8. Esse valores simbolizam que um *cluster* só deve ser considerado "sobrecarregado" se a porcentagem média da utilização for superior a 60% dos recursos disponíveis na máquina.

5.3 Considerações

O capítulo detalhou as funções que foram inseridas ou modificadas no projeto original do SPEED para o acréscimo dos algoritmos de detecção, migração dos *hot-peer* e de balanceamento completo propostos por essa monografia. Apresentou também os testes de desempenho comparando com trabalhos anteriores e concluiu a eficácia dos métodos.

O próximo e último capítulo irá concluir de maneira geral essa monografia e sugerir modificações para trabalhos futuros.

6. Conclusão e trabalhos futuros

O SPEED [1] é um projeto que alia as características de um PDMS com conceitos de semântica e, por sua estrutura distribuída singular composta de *clusters* semânticos, precisa ser balanceado para obter os melhores resultados nas consultas. Houveram diversos trabalhos discutindo os critérios de desbalanceamento em softwares que seguem estruturas semelhantes ao do SPEED [20,12,19]. Esse trabalho analisou os limiares de carga máximo (TC_{MAX}) e carga mínima (TC_{MIN}), os limiares de similaridade *cluster threshold* e *neighbor threshold* e utilizou esses limiares para identificar se há um desbalanceamento. Analisou também as operações existentes para o balanceamento semântico e de cargas. Foram detalhadas as operações de divisão, união e migração de *peers* em *clusters* semânticos e pequenas modificações nas operações foram realizadas para que se adequasse melhor ao projeto.

Essa dissertação identificou que muito dos algoritmos de balanceamento para o SPEED assumiam a existência de um *hot-peer* na rede, mas não havia uma descrição clara de como identificá-lo. Para solucionar esse problema, o algoritmo de detecção de *hot-peer* foi proposto. O algoritmo utiliza critérios como o custo da consulta para saber a probabilidade de um determinado *peer* ser o *hot-peer* do *cluster* e, portanto, identificá-lo como tal. O custo associado à operação de detecção também foi estudado e descobriu-se que varia linearmente com o número de *peers* em um mesmo *cluster*.

As operações de balanceamento foram modificadas para utilizar-se do algoritmo de detecção de *hot-peer* e um algoritmo geral de balanceamento foi proposto. O algoritmo de balanceamento geral foi avaliado quanto ao tempo de execução, grau de relevância para o balanceamento semântico e desempenho em relação ao tempo de consulta. Os experimentos desse algoritmo mostrou muito bons resultados havendo uma redução considerável no tempo médio de consulta.

Como trabalhos futuros novas formas de detecção do *hot-peer* e de balanceamento podem ser propostas. Uma métrica a ser explorada é a qualidade da rede: trabalhos futuros poderiam usar características de banda para reorganizar os *cluster*. Outra sugestão seria testar os algoritmos

de balanceamento em uma rede de diversos computadores heterogêneos e tentar identificar o impacto para a solução.

Trabalhos futuros podem ainda tentar minimizar a utilização de parâmetros. Idealmente os limiares que atualmente são definidos pelo usuário poderiam ser estudados como um problema de otimização e a melhor configuração ser fornecida diretamente pelo sistema.

7. Referências Bibliográficas

[33] Andreza Leite Alencar and Ana Carolina Salgado, "A Visual Query Interface for Ontology-based Peer Data Management Systems," *Anais do SBSI*, pp. 1-8, 2013.

[8] B Pourebrahimi, K. Bertels, and Vassiliadis S., "A survey of peer-to-peer networks," in *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Proessing*., 2005.

[1] C. E. S Pires, *Ontology-based Clustering in a Peer Data Management System*. Recife, PE, Brazil : PhD Thesis, Federal University of Pernambuco, 2009.

[14] C. E. S. Pires, B. F. Lóscio, and A. C Salgado, "Data Management in P2P Systems," *In Proc. of the 21o Simpósio Brasileiro de Banco de Dados*, 2006.

[18] C. E. S. Pires, D. Souza, T. Pachêco, and A. C. Salgado, "SemMatcher: A Tool for Matching Ontology-based Schemas. In: 24th Brazilian Symposium on Data Bases," *SBBD'09*, 2009.

[22] D.D.Gatade Vishakha Patange, "Survey of Load Balancing Approachs in Peer-To-Peer Network," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 3, no. 2, May 2012.

[4] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati, "Data management in peer-to-peer data integration systems," *Global Data Management*, pp. 177-201, 2006.

[35] Eclipse. (2014, June) [Online]. http://www.eclipse.org/downloads/packages/eclipse-standard-44/lunar

[5] Edemberg Rocha Silva, Bernadette Farias Lóscio, and Ana Carolina Salgado, "A Proposal to Maintain the Semantic Balance in Cluster-based Data Integration Systems,"

in *ICEIS*, 2014.

[6] Edemberg Rocha Silva, Guilherme Barros de Souza, and Ana Carolina Salgado, "Load Balance for Semantic Cluster-based Data Integration Systems," in *Proceedings of the 17th International Database Engineering - Applications Symposium*. Barcelona, Spain: ACM, 2013, pp. 174-179.

[29] Edemberg Rocha Silva, *Manutenção de Clusters Semânticos em Sistemas de Integração de Dados em Ambientes P2P*. Recife, PE, Brazil: PhD Thesis, Federal University of Pernambuco, 2014.

[2] G. Madhu, Dr. A. Govardhan, and Dr. TV Rajinikanth, "Intelligent Semantic Web Search Engines: A Brief Survey," *International journal of Web & Semantic Technology* , vol. 2, no. 1, 2011.

[10] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world ," in *Proceedings of the 4th USENIX*, Seatle, WA, 2003, pp. 127-140.

[7] Guilherme Barros Souza, *Operações de Balanceamento Semântico e de Carga nos Clusters do SPEED*. Recife, Brazil: Master Thesis Cin UFPE, 2012.

[12] H. V. Jagadish, B. C. Ooi, M. Rinard, and Q. H. Vu, "BATON: A Balanced Tree Structure for Peer-to-Peer Networks," in *Proc. of the 31st VLDB Conference*, Trondheim, Norway , 2005.

[13] J. Aspnes, J. Kirsch, and A. Krishnamurthy, "Load Balancing and Locality in Range- Queriable Data Structures," in *PODC'04*, St. Johns, Newfoundland, Canada, 2004.

[24] J. Garofalakis and T. Michail, "A Load Balancing in a Cluster-Based P2P System.," *Proceedings of Fourth Balkan Conference in Informatics*, pp. 133-138, 2009.

[15] K. Hose, C. Lemke, and K. Sattler, "Processing Relaxed Skylines in PDMS Using Distributed Data Summaries," *CIKM'06*, 2006.

[9] Kostas Stefanidis, "A Survey of Peer-to-Peer Systems," *Kafura*, 2009.

[28] LG Alex Sung et al., "A survey of data management in peer-to-peer systems," *School of Computer Science, University of Waterloo*, 2005.

[17] Li Ding, Pranam Kolari, Zhongli Ding, Sasikanth Avancha, and Anupam Joshi, "Using Ontologies in the Semantic Web: A Survey," in *TR-CS-05-07.*, 2005.

[3] Micha Ober, Stefan Katzenbeisser, and Kay. Hamacher, "Structure and anonymity of the bitcoin transaction graph," *Future Internet*, vol. 5, no. 2, pp. 237-250, 2013.

[34] Oracle. (2014, June) [Online]. http://www.oracle.com/technetwork/java/javase/overview/index.html

[20] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems," in *Proceedings of the 30th VLDB Conference*, Toronto, Canada, 2004.

[31] P. Obermeier and L Nixon, "A Cost Model for Querying Distributed RDF-Repositories with SPARQL," *In Proceedings of the Workshop on Advancing Reasoning on the Web: Scalability and Commonsense.*, vol. 350, 2008.

[36] P. Raftopoulou and E. G. M Petrakis, "A Measure for Cluster Cohesion in Semantic Overlay Semantic," *Proceedings of the 2008 ACM workshop on Large- Scale distributed systems for information retrieval*, 2008.

[25] P., Xiruhaki, C., Koubarakis, M., and Ntarmos Triantaffilou, "Towards High Performance Peer-to-Peer Content and Resource Sharing Systems ," *Proceedings of CIDR*, 2013.

[16] R. Hayek, G. Raschia, P. Valduriez, and N. Mouaddib, "Data Sharing in P2P Systems. Handbook of Peer-to-Peer Networking ," *LLC*, vol. 4, pp. 531-570, 2009.

[11] S. Wang, Q. H. Vu, B. C. Ooi, A. K. H. Tung, and L. Xu, "Skyframe: a framework for skyline query processing in peer-to-peer systems, ," *The VLDB Journal*, vol. 18, pp. 345-362, 2009.

[23] S., Ayyasamy and S. Sivanandam, "A Cluster Based Replication Architecture for Load Balancing in Peer-to-Peer Content Distribution.," *International Journal of Computer Networks & Communications (IJCNC)*, vol. 2, pp. 158-172, 2010.

[21] Souza D., T. ARRUDA, A. C. Salgado, P. C. A. R. Tedesco, and Z. Kedad, "Using Semantics to Enhance Query Reformulation in Dynamic Environments," *Proceedings of the 13th ADBIS*, pp. 78-92, 2009.

[32] Thiago Arruda, Damires Souza, and Ana Carolina Salgado, "PSemRef: Personalized Query Reformulation Based on User Preferences," *Proceedings of the 12th International Conference on Information Integration and Web-based Applications' Services*, pp. 683-686, 2010.

[27] Vidal Martins, Esther Pacitti, Patrick Valduriez, and others, "Survey of data replication in P2P systems," 2006.

[30] Weiyi Meng, Chengwen Liu, Wei Sun, and Clement Yu, "Predict query processing cost in a distributed database system," in *Database and Expert Systems Applications*., 1993, vol. 720, pp. 122-133.

[26] Yasushi Saito and Marc Shapiro, "Optimistic Replication," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 42-81, March 2005.

[19] Ying Qiao and Gregor von Bochmann, "Applying diffusive load balancing schemes in a clustered P2P system.," *Stud. Inform. Univ.*, no. 8.3, pp. 99-128, 2010.